MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Sebastián Hricko

Výuka programovania

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Jan Kofroň, Ph.D.

Studijní program: Informatika

Studijní obor: Programování a softwarové systémy

Rád by som poďakoval RNDr. Janu Kofroňovi, Ph.D. za odborné vedenie, cenné rady, ústretovosť pri konzultáciách a za pomoc pri spracovávaní tejto bakalárskej práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V ..... dne..... podpis

Název práce: Výuka programovania

Autor: Sebastián Hricko

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Jan Kofroň, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: Práca si kladie za cieľ vytvoriť mobilnú aplikáciu pre operačný systém Android, ktorá slúži na výuku programovania. Na rozdiel od podobných už existujúcich aplikácií sa nezameriava primárne na výuku konkrétneho programovacieho jazyka, ale na pochopenie dôležitých konceptov, ktoré sú spoločné pre väčšinu jazykov. Aplikácia, okrem výuky, obsahuje aj testovú časť, v ktorej si môže užívateľ vyskúšať správne porozumenie konceptov. Vďaka tomu, po osvojení si prebratých konceptov, bude jednoduchšie pre užívateľa prejsť na konkrétny jazyk a bude schopný, po naučení sa syntaxe, vytvárať jednoduché programy.

Klíčová slova: Android, aplikace, výuka, programování, jazyk

Title: Programming Lessons

Author: Sebastián Hricko

Department: Department of Distributed and Dependable Systems

Supervisor: RNDr. Jan Kofroň, Ph.D., Department of Distributed and Dependable Systems

Abstract: This thesis aims to create a mobile app for the Android operating system, used for teaching programming. Unlike similar already existing applications, it is not intended primarily on learning a particular programming language, but to understand the important concepts that are common to most languages. In addition to teaching, the application also includes a test section, where the user can check whether he understand concepts properly. This will make it easier for a user to switch to a specific language and to be able to create simple programs after learning the syntax.

Keywords: Android, application, lessons, programming, language

Obsah

1. Úvod

Mobilné zariadenia sa v dnešnej dobe stali bežnou súčasťou každodenného života. Takzvané smartfóny má dnes takmer každý, hlavne kvôli ich dostupnosti a prínosu pre každodenný život. Jedným z využití smartfónov je získavanie informácií a učenie sa nových vecí.

Výhodou učenia sa pomocou mobilného zariadenia je dostupnosť, ktorá je v určitých ohľadoch lepšia, než dostupnosť počítačov. Vďaka tomu, že si mobilné telefóny berieme všade so sebou, je jednoduchšie si, napríklad počas krátkej cesty autobusom, zapnúť aplikáciu, prečítať zopár strán, alebo vyriešiť jednu – dve menšie úlohy. To ma motivovalo k vytvoreniu aplikácie pre mobilné zariadenia, pomocou ktorej budú môcť užívatelia získavať vedomosti kdekoľvek a kedykoľvek.

Táto práca má za cieľ sa odlíšiť od existujúcich aplikácií na výuku programovania tým, že učí koncepty nezávisle na konkrétnom programovacom jazyku. U väčšiny podobných aplikácií platí, že prioritou je výuka konkrétneho jazyka a teda syntaxe. Na rozdiel od toho, programovací jazyk sa v tomto projekte bude využívať ako prostriedok výuky, nie ako cieľ. Ďalším rozdielom je jazyk aplikácií, kde u väčšiny prevláda angličtina. Na jednej strane je použitie angličtiny výhodné, pretože sa tým môže zvýšiť dosah a aplikácia má tak potenciál dostať sa k väčšiemu počtu užívateľov. Avšak negatívom ostáva fakt, že anglický jazyk ešte stále nie je dostatočne dobre ovládaný populáciou na Slovensku a v Českej republike. Vyplýva to z dát prieskumu jazykových kompetencií z roku 2012 organizovaný Európskou Komisiou [1], z ktorých zisťujeme, že anglický jazyk ovláda na Slovensku 12,8% respondentov a v Českej republike 11.75% opýtaných účastníkov prieskumu.

Na druhej strane sa v poslednej dobe zvyšuje povedomie o potrebnosti takzvaného informatického myslenia, ktoré sa začína vštepovať už deťom na základných školách. Vznikajú stále nové iniciatívy snažiace sa zmeniť prístup k výučbe programovania. Príkladom buď rozhovor s prof. RNDr. Ivanom Kalašom, PhD [2], v ktorom je predstavený zámer projektu IT Akadémia, ktorým je zmena prístupu k výučbe programovania. *„Školská informatika, akú sa snažíme na Slovensku rozvíjať, nechce byť náukou o počítačoch, ani náukou o programovaní, nechceme, aby podporovala technokratický pohľad na úlohu digitálnych technológií vo vzdelávaní (kedy cieľom vzdelávania je sama technológia). V našej vízii je informatika a jej jazyk – teda programovanie – nástrojom na skúmanie sveta.“ (Kalaš, 2017).*

Aplikácia si dáva za cieľ uviesť človeka do sveta programovania a poskytnúť mu základy informatického myslenia. Pri výuke sa používajú paralely s reálnym svetom, ktoré zjednodušujú pochopenie, pretože sa odkazujú na skutočnosti, s ktorými sa človek už niekedy stretol.

Text práce je rozčlenený do niekoľkých častí. Najprv si v časti Analýza ozrejmíme priebeh vytvárania aplikácie a rozoberieme si všetky dôležité

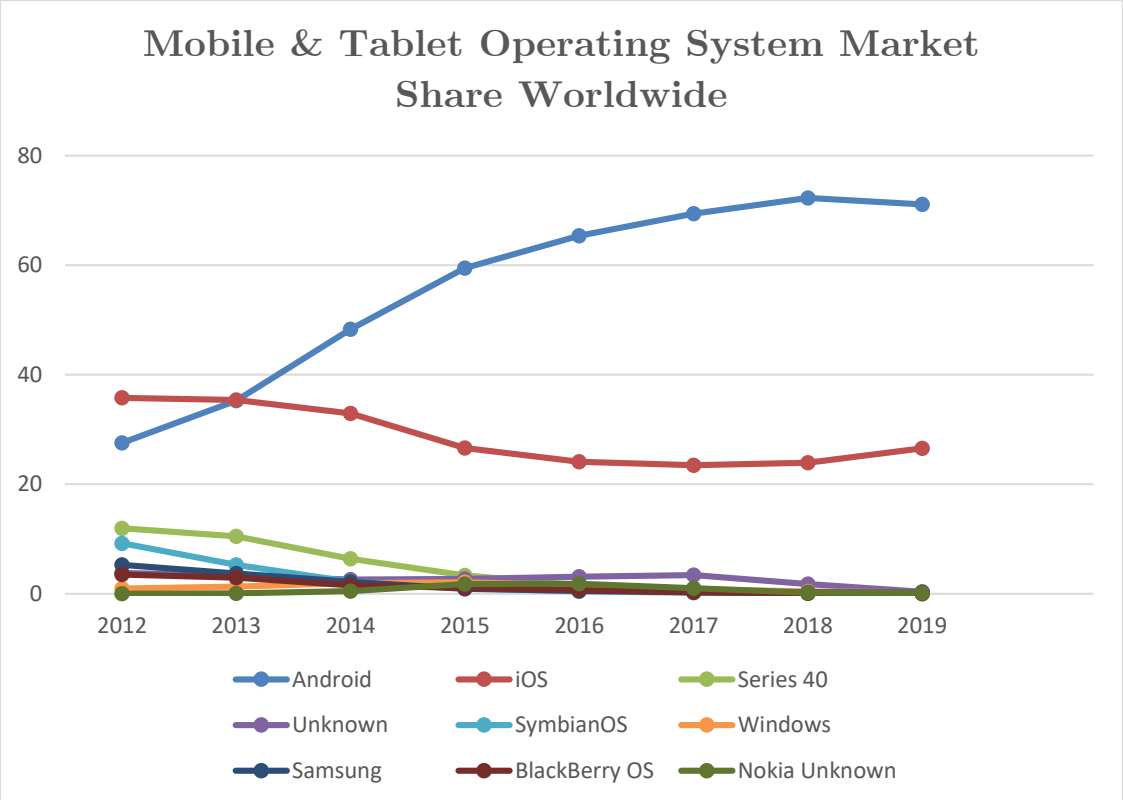
rozhodnutia, ktoré ovplyvnili výslednú prácu. V kapitole Popis aplikácie si ukážeme základné implementačné detaily všetkých dôležitých častí aplikácie. Ďalšia kapitola, Popis prekladača, je venovaná modulu interpreter, ktorý sa využíva na spustenie kódu. Popíšeme si v nej gramatiku jazyka, vysvetlíme si syntax a pozrieme sa na príklad použitia. Napokon v poslednej časti, Spôsob výuky, ozrejmíme postupy a princípy, podľa ktorých sa vytvárali jednotlivé výukové texty.

2. Analýza

Táto kapitola pojednáva o rozhodnutiach uskutočnených v priebehu vývoja práce a dôvody, kvôli ktorým boli jednotlivé technológie zvolené. Tieto technológie si stručne popíšeme.

2.1 Platforma

Vzhľadom k tomu, že vytvárame mobilnú aplikáciu, máme na výber z niekoľkých operačných systémov. Nahliadnutím do grafu vyhotoveného z dát spoločnosti StatCounter GlobalStats môžeme zistiť, že najpopulárnejším operačným systémom pre mobilné telefóny a tablety je operačný systém Android, za ním nasleduje iOS. Ďalšie operačné systémy majú tak nízky počet užívateľov, že vývoj pre tieto platformy je zbytočný (Viď Obrázok 2.1: Celosvetový trhový podiel operačných systémov pre mobilné telefóny a tablety).



Obrázok 2.1: Celosvetový trhový podiel operačných systémov pre mobilné telefóny a tablety [3]

Tento podiel na trhu bol jedným z hlavných kritérií podľa ktorých som sa rozhodoval, aký operačný systém zvolím. Vďaka tomu má aplikácia potenciál dostať sa k väčšiemu množstvu užívateľov.

Ďalším z kritérií bola aj moja skúsenosť s vývojom aplikácií pre Android. Už pri začiatku vývoja som mal povedomie o technológiách, ktoré umožňuje tento operačný systém používať, na rozdiel od konkurenčného systému iOS.

Posledným dôvodom výberu platformy Android je fakt, že štandardný vývoj aplikácií pre iOS je podmienený počítačom od Applu, v ktorom by bolo možné spustiť vývojové prostredie Xcode používané pri vývoji aplikácií. Android ale špeciálne zariadenie nepotrebuje a vývojové prostredie Android Studio je možné spustiť na počítačoch s operačným systémom Microsoft Windows, Mac OS alebo Linux.

Na druhej strane za veľkú nevýhodu Androidu považujem vysoké množstvo výrobcov zariadení, ktoré využívajú OS Android. Tieto zariadenia disponujú rozlične veľkými obrazovkami, verziami systému a rôznymi nadstavbami od výrobcu, ktoré značne spomaľujú vývoj aplikácií pre tieto zariadenia. Naproti tomu, vo svete iOS, tento problém nie je natoľko významný vďaka faktu, že jednotlivé zariadenia sa od seba nelíšia zásadným spôsobom tak, ako je tomu u zariadení s operačným systémom Android. Tento problém čiastočne rieši prítomnosť emulátoru zariadení Android v programe Android Studio, vďaka ktorému je možné nájsť a odladiť väčšinu chýb spojených s rozdielnymi veľkosťami obrazoviek. Emulátor taktiež do istej miery rieši problémy týkajúce sa verzií operačného systému, ktoré by vývojár nebol schopný odladiť iba s pomocou jediného fyzického zariadenia. Naďalej však ostávajú problémy s nadstavbami operačného systému od výrobcu, ktoré môžu ovplyvniť spôsob behu aplikácií na pozadí, prideľovanie prostriedkov pre aplikácie a ďalšie skutočnosti, s ktorými musí vývojár pri vytváraní aplikácií počítať.

2.1.1 Podpora verzií Androidu

Po výbere operačného systému nastal čas na rozhodnutie sa pre minimálnu verziu systému Android, ktorú bude aplikácia podporovať. Pre lepší prehľad sa pozrime na znázornenie aktuálnych distribúcií verzií Androidu (Tabuľka 2.1).

Pri výbere verzie je nutné urobiť kompromis medzi pokrytím najväčšieho množstva zariadením a funkcionalitami, ktoré ponúkajú vyššie verzie API. V rámci mojej firemnej praxe som sa utvrdil v názore, že je užitočnejšie, ak aplikácia pokrýva pokiaľ možno čo najväčšie spektrum zariadení, pričom k najnovším funkcionalitám sa do istej miery dajú nájsť náhrady pre nižšie verzie. Aj v takých prípadoch je však nutné postupovať opatrne a ak je už

spočiatku jasné, že budeme využívať technológie dostupné iba vo vyšších API verziách, nemá zmysel podporovať nižšie verzie.

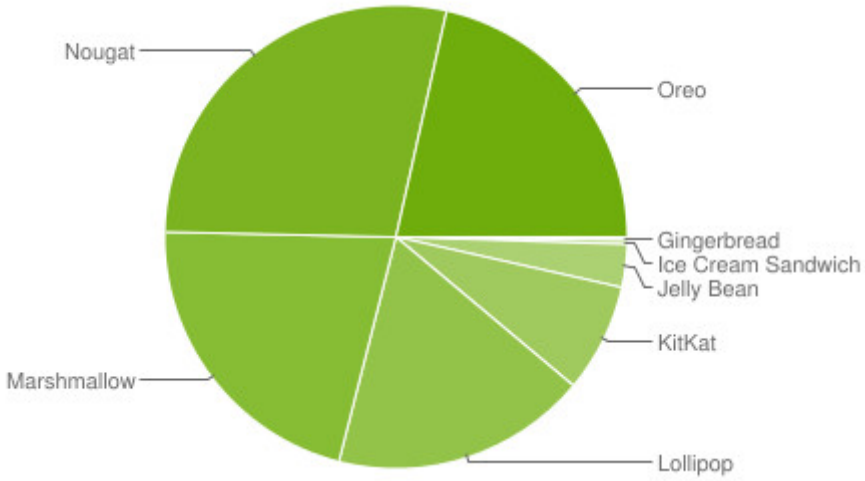
V prípade tejto práce bolo rozhodnutie jednoznačné, všetky technológie, ktoré som plánoval použiť fungujú bez problémov na API verzii 19 a vyššej. To umožňuje – podľa grafu (Obrázok 2.2) – fungovanie až na 96.5% zariadení, čo je dostatočné pre účely tejto aplikácie.

Po stanovení minimálnej verzie API bolo ešte potrebné rozhodnúť sa pre *targetSdkVersion* (cieľová verzia) a *compileSdkVersion* (verzia pre preklad). Obe sú v projekte nastavené na najvyššiu verziu API, ktorá je v dobe písania textu 28[1]. Je to z dôvodu odporúčania zo strany Googlu [4].

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.2 %
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3 %
4.1.x	Jelly Bean	16	1.1 %
4.2.x		17	1.5 %
4.3		18	0.4 %
4.4	KitKat	19	7.6 %
5.0	Lollipop	21	3.5 %
5.1		22	14.4 %
6.0	Marshmallow	23	21.3 %
7.0	Nougat	24	18.1 %
7.1		25	10.1 %
8.0	Oreo	26	14.0 %
8.1		27	7.5 %

Tabuľka 2.1: Podrobné informácie o distribúcií verzií Androidu prevzaté z webu Google Developers [5]

[1] V uvedených diagramoch táto verzia nie je zahrnutá z dôvodu zberu dát pred uvedením Androidu 9 do produkcie.



Obrázok 2.2: Diagram distribúcie verzií Androidu podľa dát z webu Google Developers [5]

2.1.2 Programovací jazyk

Pri výbere programovacieho jazyka znovu zohrávali úlohu skúsenosti s konkrétnymi jazykmi. V dnešnej dobe existuje podpora pre jazyk Java, Kotlin, C# (Xamarin), Python, C++ a HTML5.

Pomocou jazyka HTML5 sa vytvárajú takzvané hybridné aplikácie, ktoré bežia na mobilných telefónoch, ale využívajú webové technológie. Nedá sa takto vytvárať natívne aplikácie, preto tento jazyk nepripadá do úvahy.

Jazyk C++ slúži na vytváranie Android knižníc, ktoré fungujú na backende, čo účelom projektu nevyhovuje. S týmto je spojený aj jazyk Python, ktorý sa používa na podobné účely ako jazyk C++ a preto taktiež nie je vhodný pre vývoj aplikácie s užívateľským rozhraním.

C#, ktorý s pomocou Xamarinu dovoľuje vývojárom vytvárať natívne aplikácie pre Android a iOS zároveň, som taktiež zavrhol kvôli malej komunite vývojárov a nízkemu počtu open source knižníc.

Ako posledné sú na výber jazyky Kotlin a Java. Oba tieto jazyky sú oficiálne podporované jazyky na vývoj mobilných aplikácií pre Android. Kotlin je moderný jazyk, ktorý vyvinula spoločnosť JetBrains a vydaný bol 15. februára 2016 [6]. Oficiálne podporovaným sa stal na konferencii Google I/O v roku 2017 [7]. Odvtedy sa začal vo veľkej miere používať na vývoj Android aplikácií ako programovací jazyk, ktorý má byť jednoduchší a prehľadnejší. Za pomerne krátky čas sa vytvorila dostatočne veľká komunita vývojárov, ktorí tento jazyk používajú a preto už nie je výnimkou nájsť množstvo tutoriálov v Kotline. Medzi špecifiká Kotlinu patrí skutočnosť, že od základov sa počíta

napríklad s konceptom null safety a smart cast, čím sa odlišuje od Javy. Pri rozhodovaní sa som dal však prednosť Jave, ktorá ešte stále prevažuje nad Kotlinom v prípadoch komunity a podpory, ktorá je za tie roky obrovská.

2.2 Koncepty

Ďalším z rozhodnutí, ktoré bolo potreba uskutočniť, je spôsob, akým sa ukladajú dáta o jednotlivých konceptoch. V tomto prípade som uvažoval nad dvoma možnosťami: JSON súbor, alebo SQLite databáza. Výhodou SQLite databázy je fakt, že je priamo podporovaná Androidom a z toho dôvodu je používanie veľmi dobre optimalizované. Vzhľadom k tomu, že dáta o konceptoch budú statické a nebude potreba ich meniť dynamicky za behu aplikácie, ukázalo sa, že použite JSON súboru je na tieto účely postačujúce.

Pre SQLite databázu by som sa rozhodol v prípade, ak bol súbor s konceptmi veľký a nebolo by možné ho celý uložiť do pamäte ako objekty. V prípade tohto projektu sa však jedná o súbor veľkosti (v dobe písania textu), ktorá sa pohybuje okolo 40 kB.

Na parsovanie dát zo súboru vo formáte JSON bola použitá open-source knižnica Gson vyvíjaná spoločnosťou Google ako jednoduchý a rýchly nástroj na prevod medzi objektmi a reťazcom vo formáte JSON. Tento spôsob bol zvolený primárne na základe jednoduchej použiteľnosti API, ktoré poskytlo veľmi prehľadný a krátky kód bez potreby vytvárania vlastných adaptérov. Ďalším faktorom bola rýchlosť, ktorá je oproti konkurenčným nástrojom spravidla lepšia až priemerná [8].

2.2.1 Výuka

Od počiatku bolo jasné, že výuka bude obsahovať množstvo textu a bude doplnená obrázkami, vývojovými diagramami. Preto som hľadal spôsob, ktorým v aplikácii zobraziť tieto texty tak, aby ich čítanie bolo dostatočne zážívné pre koncového užívateľa.

Jedna z možností, ktorú som bral do úvahy bolo rozdelenie každej výuky na samostatné stránky použitím komponenty ViewPager. Každá stránka by pritom obsahovala v záhlaví obrázok, ktorý by znázorňoval to, čo obsahuje daný výukový text. Samotný text by bol potom v tele stránky.

Komplikácie spojené s týmto prístupom však nastali v prípade, keď som nechcel jeden text rozdeľovať na viacero obrazoviek a zároveň som potreboval znázorniť obrázky do tela obrazovky. Keďže jedna obrazovka mala obrázok v záhlaví a text v tele, neumožňoval mi takýto návrh zobrazovanie väčšieho počtu obrázkov na jednej obrazovke.

Ďalším problémom, ktorý sa s tým spája, bolo taktiež zobrazovanie vývojových diagramov, ktoré som vo veľkej miere využíval vo výuke. Záhlavie muselo byť dostatočne malé na to, aby nenarúšalo celkový dizajn obrazoviek a preto sa diagramy do týchto záhlaví nevošli. Výhodou tohto prístupu by bol však fakt, že všetok text výuky by mohol byť obsiahnutý v dokumente formátu JSON, ktorý by obsahoval taktiež referencie na obrázky uložené v aplikácii.

Aj kvôli uvedeným problémom som sa rozhodol pre vytvorenie samostatného súboru vo formáte PDF pre každý text výuky. Tento spôsob mi umožňoval vytvoriť dizajn, ktorý obsahuje text a zároveň obrázky. Vyriešil sa tým taktiež problém so zobrazovaním diagramov, pretože návrh jednotlivých obrazoviek nebol porušený. Každý výukový text v aplikácii je rozdelený na stránky, ktoré reprezentujú stránky súboru PDF.

2.2.2 Testy

V aplikácii sú dva druhy testov, prvým druhom sú otázky s odpoveďami vo forme štyroch možností, z ktorých je správna jedna, alebo viac. Druhý spôsob odpovede na príslušné otázky je napísanie niekoľkých riadkov kódu, ktoré sa následne vyhodnotia.

Všetky potrebné údaje o testoch sú zahrnuté vo formáte JSON priamo v súbore, ktorý obsahuje všetky koncepty. Takéto použitie sa ukázalo ako postačujúce, pretože všetky otázky mali podobný charakter. Pre oba druhy je vytvorený samostatný layout, ktorý je možné jednoducho naplniť príslušnými otázkami v správnom formáte.

2.3 Interpreter

Kvôli vyhodnocovaniu testov bolo nutné vytvoriť nástroj, ktorý bude schopný preložiť, spustiť a vyhodnotiť napísaný kód. Bolo potrebné nájsť spôsob, ktorým by bolo možné nad kódom spustiť lexikálnu, syntaktickú a sémantickú analýzu.

2.3.1 Generátor prekladača

Na tieto účely sa používajú takzvané generátory prekladačov, pričom sa vytvára syntaktický analyzátor a v našom prípade interpreter. V tomto projekte sa používa nástroj ANTLR [9], ktorý slúži na vytváranie analyzátora jazyka na základe uvedenej gramatiky. ANTLR je moderný a rýchly nástroj, ktorý sa od ostatných podobných nástrojov líši napríklad tým, že automaticky generuje takzvané „parse tree walkers“, ktoré dovoľujú prechádzať derivačný

strom reprezentujúci syntaktickú štruktúru zdrojového kódu na základe zadanej gramatiky.

2.3.2 Prechod derivačného stromu

Nástroj ANTLR ponúka dva spôsoby, ktorými je možné prejsť derivačný strom. Na jednej strane je tu možnosť použitia listenerov, kde použité metódy volá zabudovaný ANTLR walker pri prechode stromom. Druhý spôsob, ktorý je použitý aj v tomto projekte je postavený na báze visitora, kde prechod stromom nie je automatický a je preto nutné do jednotlivých vrcholov v strome vstupovať.

Visitor je použitý nielen z dôvodu úplnej kontroly prechodu stromom, ale taktiež kvôli spôsobu ukladania údajov počas prechodu stromom. Pri použití visitoru je možné definovať návratovú hodnotu „návštevy“. To sa líši od použitia listeneru, v ktorom tento prístup nie je možný, pretože jednotlivé metódy nevracajú hodnoty a je preto nutné ukladať výsledky do atribútov triedy daného listeneru. V prípade následnej potreby vybratia výsledku je potrebné získať hodnotu uloženú v atribúte.

Posledným dôvodom je rozsah kódu, ktorý pri spôsobe použitia listeneru býva relatívne dlhý a často menej prehľadný v porovnaní s použitím prístupu visitoru.

2.4 Užívateľské rozhranie

Dizajn aplikácie bol zvolený tak, aby bol maximálne jednoduchý a aplikácia sama bola ľahko použiteľná. Pomocou pri budovaní prostredia mi bolo nasledovanie princípov takzvaného 3 C's : Concise, Clear & Consistent (Stručné, Jasné a Konzistentné).

Stručnosť je princíp, vďaka ktorému nie je užívateľ zahltený prílišným množstvom informácií, ktoré ho odradia od používania aplikácie. V tomto konkrétnom prípade bol princíp zachovaný napríklad na hlavnej obrazovke, ktorá obsahuje zoznam konceptov, pričom každý koncept je veľmi stručne – jednou vetou – popísaný. To dáva užívateľovi prehľad o tom, čo bude obsahom daného konceptu. Samotný detail konceptu je rovnako veľmi stručný. Keďže súčasťou aplikácie je aj výuka, ktorá z podstaty veci obsahuje množstvo textu, nebolo možné implementovať princíp stručnosti na tieto texty, pretože výuka musí byť do rozumnej miery podrobná.

Ďalším princípom je snaha o zachovanie jasného a zreteľného prostredia. V aplikácii bol tento princíp použitý v dvoch rovinách. Prvou z nich je rovina zrozumiteľnosti objektov, čo znamená, že každý prvok užívateľského prostredia má svoj význam a tento význam je jasný. Ako príklad uvediem znovu hlavnú obrazovku so zoznamom konceptov. Každá karta

v zozname predstavuje jeden koncept a užívateľ očakáva, že v prípade, ak na niektorú z nich klikne, dostane sa k detailu konceptu. V rámci testovania sa ukázalo, že užívatelia boli schopní sami prejsť všetkými obrazovkami aplikácie a nenastal tak problém s nejasnosťou jednotlivých komponent. Druhou rovinou je forma písania výukových textov, pričom cieľom bolo minimalizovať nejasnosti spojené s výkladom látky. Snahou bolo obmedzenie používania predom nevysvetlených termínov, alebo náročných termínov ku ktorým existuje postačujúci ekvivalent, ktorý je začiatočníkovi bližší.

Posledným princípom, ktorý som mal v pláne v aplikácii zachovať je konzistentnosť. Tento princíp vychádza z myšlienky, podľa ktorej ľudia nemajú v láske príliš netradičný dizajn a použiteľnosť aplikácie, ktorá sa líši od toho, na čo sú zvyknutí. Na tomto princípe je postavený ekosystém iOS, ktorého systémové aplikácie sa vzhľadom líšia iba minimálne. Z toho dôvodu je systém používaný na zariadeniach od spoločnosti Apple veľmi intuitívny a užívateľa iba zriedka niečo prekvapí. Taktiež to platí aj u aplikácií tretích strán, ktoré majú k dispozícii komponenty, ktoré sa bežne používajú na tejto platforme.

V prípade Androidu je snaha o zachovanie konzistentnosti náročnejšia, vzhľadom k spomínaným nadstavbám prostredia od výrobcov, ktoré dovoľujú vo veľkej miere meniť dizajn celého prostredia. Google sa však stále snaží ovplyvňovať dizajn aplikácií tretích strán prostredníctvom jednotných dizajnových systémov. S príchodom Androidu 4.0 sa v roku 2011 [10] začal používať ako štandardný dizajnový systém dizajn nazývaný Holo. Tento systém bol nahradený systémom Material design, ktorý bol predstavený v roku 2014 [11] ako náhrada za systém Holo. Material design je aktuálny dodnes a vývojári si tento dizajn zamilovali vďaka modernému vzhľadu a dôrazu na použiteľnosť. Existuje príručka, ktorá obsahuje množstvo návodov a referenčných ukážok, ktoré pomôžu pri návrhu vzhľadu aplikácie. Na používanie komponentov odpovedajúcim Material designu slúži knižnica Android Design Support library, ktorá obsahuje vždy aktuálne komponenty na používanie v aplikáciách. Z tejto knižnice som čerpal hlavne pri zozname konceptov, kde jedna položka zoznamu je karta, ktorá bola použitá tak, aby spĺňala princípy určené práve týmto dizajnovým systémom.

2.5 Ďalšie použité knižnice

V projekte okrem už spomenutých knižníc sú použité ešte dve užitočné knižnice. Prvá, SDP library [12], slúži na vytváranie responzívneho dizajnu pridaním novej jednotky miery – sdp (scalable dp). Postavená je na myšlienke rovnakého rozloženia prvkov pre akékoľvek rozmery obrazovky. Použitie tejto knižnice sa mi osvedčilo u viacero projektov, pretože tým sa zjednodušila podpora viacerých zariadení s rôznymi veľkosťami obrazoviek.

Druhou knižnicou je knižnica na zobrazovanie PDF súborov. V procese rozhodovania sa pre správnu knižnicu som vyskúšal vstavaný nástroj v Androide, PdfRenderer, ktorý by však nepodporoval zariadenia s API nižším než 21. Nevýhodou tohto prístupu bol taktiež menej kvalitný rendering jednotlivých stránok, text a obrázky pôsobili rozmazane. Napokon som sa rozhodol pre knižnicu AndroidPdfViewer [13], ktorá ponúka veľmi jednoduchý spôsob na zobrazenie súboru PDF. Toto zobrazenie je možné si nastaviť tak, aby si užívateľ mohol zväčšiť obrazovku dvojitým kliknutím, alebo gestom pinch-to-zoom. Taktiež rendering je veľmi dobrý, aj pri vyššom priblížení sú text aj obrázky na zobrazenej stránke dostatočne ostré.

2.6 Ukladanie dát

Dáta, ktoré musia byť uložené v pamäti delíme na dve časti: užívateľský postup pri výuke a kód, ktorý si užívateľ skúšal spúšťať v komponente Pieskovisko.

Užívateľský postup vo výuke sa zaznamenáva do interného úložiska pomocou privátneho súboru aplikácie, ku ktorému iné aplikácie nemajú prístup a teda je chránený pred úpravou, alebo vymazaním. Súbor bude uložený vo formáte JSON, pretože nie je natoľko komplexný, aby bolo nutné vytvárať databázu, ktorá by zjednodušila používanie.

Na ukladanie spomínaného kódu v komponente pieskovisko budeme využívať nástroj Shared preferences. Tento prístup bol zvolený vďaka svojej jednoduchosti na ukladanie hodnôt primitívnych typov, čo naše potreby spĺňa. Princíp ukladania je založený na zápise a čítaní dát uložených v tvare kľúč-hodnota.

3. Popis aplikácie

V tejto časti si ukážeme, z akých komponent pozostáva samotná aplikácia a k čomu jednotlivé časti balíčka slúžia. Úplné implementačné detaily sú uvedené v JavaDoc dokumentácii v kóde. Taktiež si ukážeme, akým spôsobom môže byť aplikácia rozšírená a na čo musíme brať zreteľ pri rozširovaní.

3.1 Stavba projektu

Projekt v Android Studiu sa skladá z modulov. Moduly môžu byť rôznych typov, pomocou nich je možné napríklad vytvárať prídavné knižnice do projektu. Ďalšou možnosťou využitia modulov je vytvorenie aplikácie rozličné zariadenia – napríklad pre chytré hodinky postavené na operačnom systéme Wear OS.

Tento projekt pozostáva z dvoch modulov **app** a **interpreter**. V tejto kapitole si popíšeme hlavný modul app, ktorý predstavuje samotnú aplikáciu. V kapitole 4 Popis prekladača sa bližšie pozrieme na modul interpreter.

3.1.1 Zdrojové súbory

Všetky zdrojové súbory modulu app sú uložené v spoločnom balíčku s názvom *cz.cuni.mff.shricko.learningprogramming*. Tento balíček ďalej obsahuje celú štruktúru „podbalíčkov“, v ktorých sú rozdelené všetky zdrojové súbory podľa svojho druhu. Hrubá štruktúra rozdelenia je znázornená na diagrame (Obrázok 3.1: Znázornenie štruktúry hlavného balíčka modulu app).

V týchto podbalíčkoch sa nachádzajú konkrétne triedy. Pri zadeľovaní zdrojových súborov do balíčkov som sa riadil týmto popisom balíčkov:

- **activity** balíček obsahuje triedy, ktoré dedia od triedy AppCompatActivity a predstavujú základné obrazovky užívateľského rozhrania. Aktivity boli v aplikácii vytvárané tak, aby každá dôležitá obrazovka predstavovala jednu aktivitu.

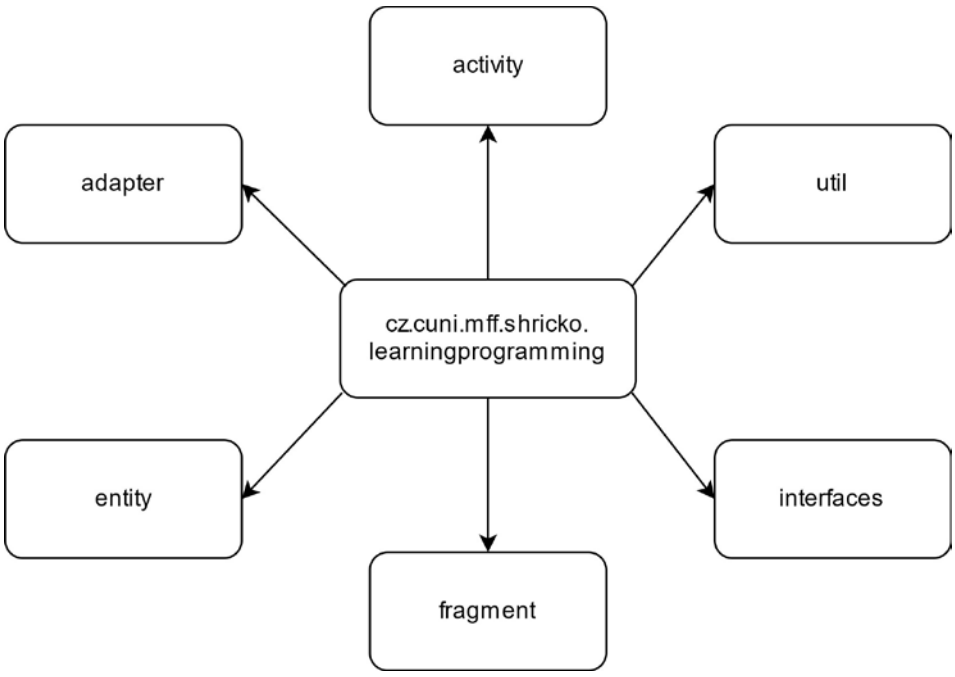
- **adapter** balíček zahŕňa triedy, ktoré slúžia na prepojenie dát s komponentou View, ktorá má byť týmito dátami naplnená. V tomto projekte sa adaptér používa na naplnenie komponenty RecyclerView, pomocou ktorej sa zobrazujú jednotlivé koncepty na hlavnej obrazovke. Adaptér umožňuje prideliť položkám ich vzhľad v samotnom zozname.

- **entity** balíček je zodpovedný za uchovávanie tried – entít vyskytujúcich sa v aplikácii. Medzi ne patrí napríklad trieda Concept predstavujúca koncept, abstraktná trieda Question, ktorá reprezentuje jednotlivé otázky v testoch.

- **fragment** balíček obsahuje triedy dediace od triedy Fragment. Fragmenty v ekosystéme Androidu sú komponenty UI, ktoré reprezentujú samostatnú znovu použiteľnú jednotku užívateľského rozhrania. Fragmenty sú v aplikácii použité v prípade užívateľských testov, pričom každá otázka v teste je zobrazená v jednom fragmente.

- **interfaces** balíček obsahuje zvláštne rozhrania (interface), ktoré sa zvyčajne používajú na komunikáciu medzi fragmentom a aktivitou.

- **util** balíček v sebe zahŕňa ostatné potrebné triedy, ktoré aplikácia na svoj beh využíva. Jednou z nich je trieda CodeRunUtil, ktorá sa používa na viacerých miestach v aplikácii a zastrešuje spúšťanie užívateľom zapísaného kódu.

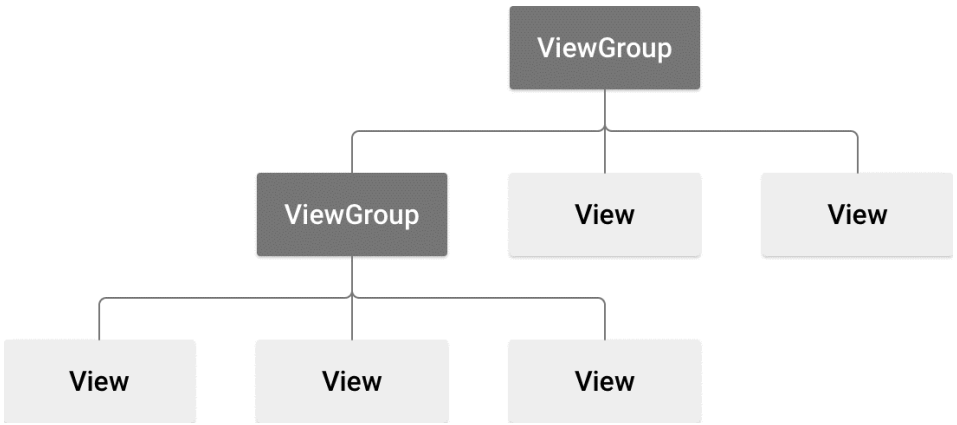


Obrázok 3.1: Znázornenie štruktúry hlavného balíčka modulu app

3.1.2 Resources súbory

Takzvané resources súbory v ekosystéme Androidu označujú prídavné súbory zahrňujúce statický obsah, ktorý predstavujú obrázky, texty, témy a štýly, animácie, súbory predstavujúce rozloženie jednotlivých obrazoviek a obrazovkových komponent a rôzne iné súbory.

Tieto súbory sú zoskupené v priečinkovej hierarchii. Priečinok layout obsahuje všetky rozloženia obrazovkových komponent a samotných obrazoviek, v prípade tohto projektu sú to súbory popisujúce rozloženie aktivít, fragmentov a položky zoznamu konceptov – súbor *carditem.xml*. Layout súbory sú vo formáte XML, pričom každý element je typu View, alebo ViewGroup (Obrázok 3.2). V týchto súboroch je vo veľkej miere využívaná komponenta ConstraintLayout, ktorá do určitej miery poskytuje dobrú responzivitu vďaka možnosti určovania rozmerových a pozičných obmedzení pre komponenty vo vnútri tejto ViewGroup komponenty. Ďalšími používanými ViewGroup komponentami v tomto projekte sú NestedScrollView, AppBarLayout a taktiež LinearLayout.



Obrázok 3.2 Ilustrácia štruktúry layout súboru. [14]

Ďalší zaujímavý resource súbor je súbor *strings.xml*, ktorý zahŕňa texty zobrazujúce sa v aplikácii. Tento spôsob ukladania textov uľahčuje lokalizáciu do iných jazykov práve vďaka oddeleniu textov od „logiky“ aplikácie. Prostredie aplikácie je v čase písania textu lokalizované do slovenčiny, ničomu však nebráni rozšírenie lokalizácie prostredia do iných jazykov.

Resource priečinok *raw* sa využíva na pripojenie prídavných súborov, ktoré môžu byť v aplikácii použité. V tomto projekte sa priečinok raw používa na uloženie vstupného súboru *concepts.json*.

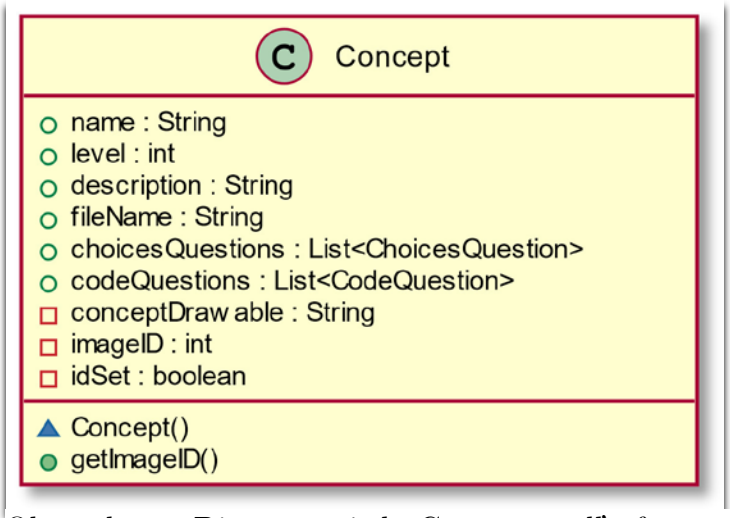
Často využívaným resource priečinkom je priečinok *drawable*, využívaný na priloženie rôznych ikon a obrázkov, využívaných v aplikácii. Takto sú,

napríklad, uložené ikony príslušných konceptov zobrazujúce sa v zozname konceptov.

Súbory výukových textov vo formáte pdf sú uložené v priečinku *assets*, do ktorého je po vytvorení nového konceptu potrebné súbor s výukovým textom vložiť. V programe sa k týmto súborom pristupuje pomocou nástroju AssetManager.

3.2 Koncept

Koncept je základná entita, okolo ktorej sa točí celá aplikácia. Návrh triedy Concept je znázornený pomocou diagramu (Obrázok 3.3: Diagram triedy Concept).



Obrázok 3.3: Diagram triedy Concept podľa formátu PlantUML

3.2.1 Načítavanie konceptov

Načítavanie konceptov prebieha pomocou vstupného súboru *concepts.json.* Súbor je prítomný v priečinku *raw* v rámci modulu app. Tento súbor obsahujúci zoznam konceptov a jednotlivé informácie o nich, je uložený vo formáte JSON. V prílohe (A.1 Vstupný JSON formát) je znázornená schéma, voči ktorej musí byť vstupný súbor validný, aby bolo načítanie nových konceptov správne. Schéma pritom poskytuje návod na zmenu, alebo vytvorenie výuky nového konceptu. Všetky atribúty okrem testov, či už s výberom možností, alebo s dopĺňaním kódu sú povinné a zaručujú správne fungovanie aplikácie.

3.2.2 Ukladanie postupu užívateľa

Načítavanie a ukladanie súboru s informáciami o postupe má na starosti trieda UserProgressUtil v balíčku *util*. Obsahuje dve statické funkcie loadProgress a saveProgress, ktoré pracujú so systémovým úložiskom a načítajú, respektíve ukladajú informácie o postupe užívateľa.

Tieto informácie sú uložené v triede UserProgress v balíčku *entity*. Trieda obsahuje dva atribúty – teachingDone a percentTest, ktoré sa používajú pri znázornení postupu užívateľa na obrazovke detailu konceptu.

3.3 Výuka

3.3.1 Výukové texty

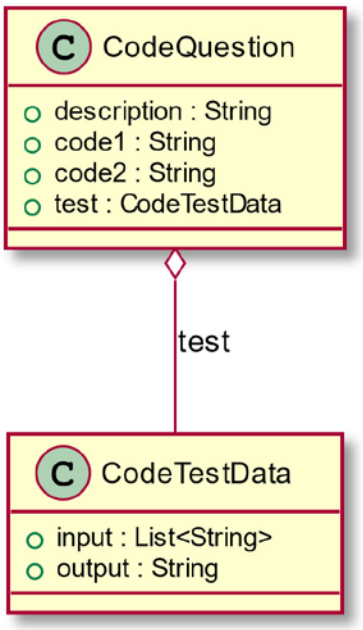
Jednotlivé výukové texty sú uložené v module app v zložke *assets*, vo formáte PDF. Názov súboru pre príslušný koncept je potom zaznamenaný vo vstupnom súbore *concepts.json* pod povinným atribútom filename.

Výukové texty sa zobrazujú v aplikácii vďaka aktivite PdfActivity.
Pri spustenie tejto aktivity máme na výber z dvoch možností. V prípade, že chceme zobraziť výuku pre koncept, je nutné tento koncept vložiť do predávaného intentu. Do intentu sa vkladá však iba pozícia konceptu v zozname všetkých konceptov.

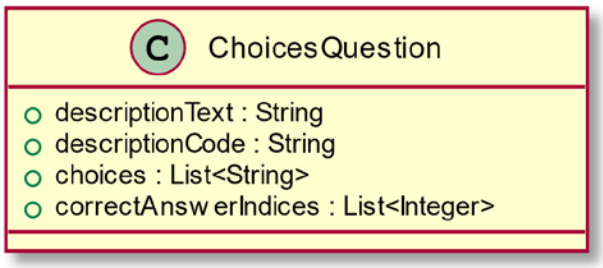
Na druhej strane existuje možnosť predať iba názov PDF súboru, ktorý sa má zo zložky *assets* otvoriť. Tento spôsob sa používa pri otvorení syntaktickej príručky.

3.3.2 Testy

Testy sa do aplikácie predávajú rovnako v rámci vstupného súboru *concepts.json*. Aplikácia pozná dva druhy otázok a to CodeQuestion (Obrázok 3.4: Diagram triedy CodeQuestion) a ChoicesQuestion (Obrázok 3.5).



Obrázok 3.4: Diagram triedy CodeQuestion



Obrázok 3.5: Diagram triedy ChoicesQuestion

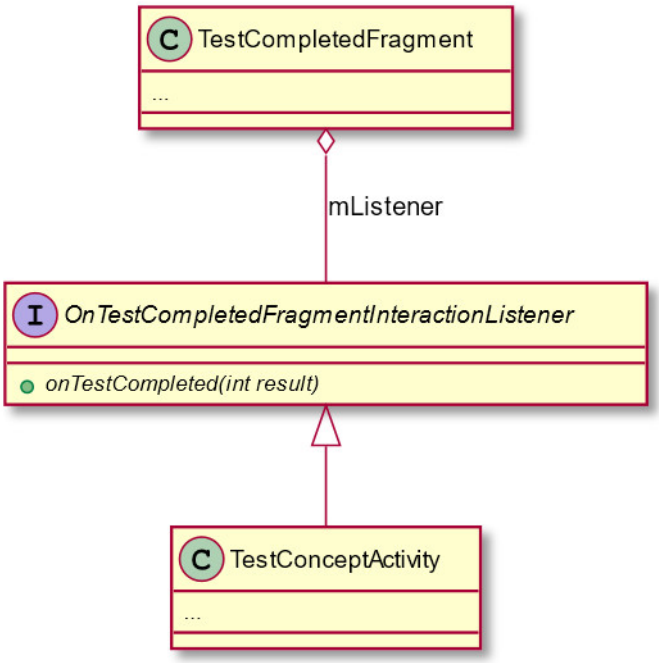
Atribúty code1 a code2 v triede CodeQuestion slúžia na ohraničenie bloku kódu, ktorý má užívateľ vložiť ako odpoveď. Výsledný kód, ktorý sa bude spúšťať bude zložený z kódu code1 + užívateľský vstup + code2. Ďalším zaujímavým atribútom je objekt triedy TestData, pričom z názvu vyplýva, že reprezentuje testovacie dáta, voči ktorým sa bude vyhodnocovať napísaný kód.

Trieda CodeQuestion obsahuje dva atribúty na popis otázky, zoznam možností, z ktorých si užívateľ môže vybrať a napokon zoznam indexov správnych možností v zozname choices.

3.4 Špecifiká komunikácie fragment - activity

Táto kapitola popisuje spôsob komunikácie medzi aktivitou a fragmentom. Vysvetľuje použite rozhraní – interface – na účely listeneru ako spôsobu podávania informácií.

Použitie si ukážeme na príklade komunikácie medzi triedou TestConceptActivity a triedou TestCompletedFragment. Druhá zo spomínaných vyžaduje od aktivity, ktorá je volacou, aby implementovala interface OnTestCompletedFragmentInteractionListener. Diagram je zobrazený na obrázku.



Obrázok 3.6 Znázornenie komunikačného rozhrania medzi fragmentom a aktivitou

Na uvedenom diagrame je prostrednou komponentou interface, ktorý obsahuje metódu onTestCompleted(int result). Táto metóda je konkrétne implementovaná v aktivite nasledujúcim spôsobom:

```
@Override
public void onTestCompleted(int result) {
   AllConcepts allConcepts = AllConcepts.getInstance(this);
   if (concept != null && allConcepts.getProgressTest(concept.name) < result)
     AllConcepts.getInstance(this).setProgress(concept.name, result, this);
}
```

Použitie tejto metódy je v triede TestCompletedFragment, kde sa interface priradzuje do atribútu mListener. Toto priradzovanie prebieha v metóde fragmentu onAttach(Activity activity), ktorá sa volá hneď pri zavolaní fragmentu príslušnou aktivitou a teda pri pripojení fragmentu k aktivite. Odobratie listeneru je vykonané v metóde onDetach(), ktorá je zavolaná pri odpojení fragmentu od aktivity ako posledná udalosť v životnosti fragmentu. Priradenie a odobratie listeneru do/z atribútu mListener prebieha nasledovne:

```
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    try {
        mListener = (OnTestCompletedFragmentInteractionListener) context;
    } catch (ClassCastException e) {
        throw new ClassCastException(context.toString() + "must implement" +
        OnTestCompletedFragmentInteractionListener.class.getSimpleName());
    }
}

@Override
public void onDetach(Context context) {
    super.onAttach(context);
    super.onDetach();
    mListener = null;
}
```

Nakoniec môžeme metódu onTestCompleted(int result) na tomto objekte použiť vo fragmente kedykoľvek bude použitie nutné. V našom prípade bola metóda použitá pri vytvorení fragmentu a zobrazení percent, ktoré sa použitím tejto metódy uložia do objektu triedy Concept.

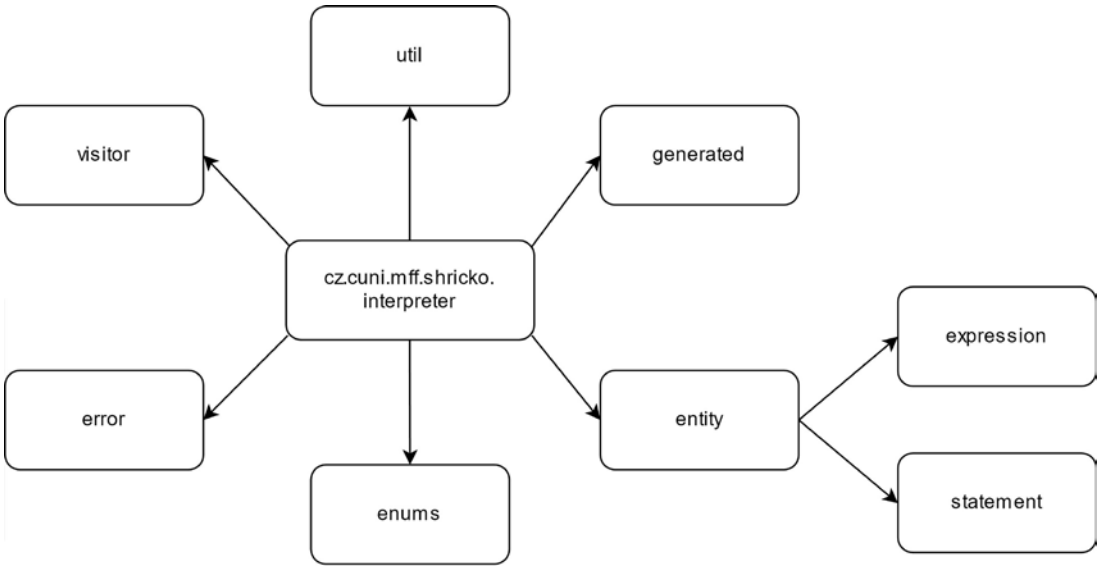
4. Popis prekladača

V tejto kapitole sa pozrieme na modul interpreter, ktorý predstavuje knižnicu na spúšťanie vlastného kódu pomocou generátora prekladača ANTLR. Ďalej si v kapitole ukážeme aj štruktúru vlastného jazyka a príklad programu zapísaného v tomto jazyku.

4.1 Stavba modulu

Modul interpreter je vytvorený ako knižnica v jazyku Java. Je preto možné použiť ju v iných projektoch, nielen v aplikáciách pre operačný systém Android. Modul tvorí hlavný balíček *cz.cuni.mff.shricko.interpeter.* Ten sa ďalej delí na podbalíčky, ktoré obsahujú triedy, podľa svojho druhu.

Schéma balíčka je znázornená na obrázku (Obrázok 4.1).

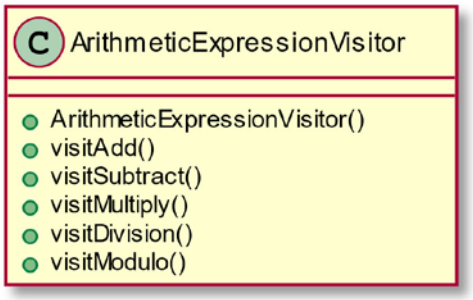


Obrázok 4.1: Znázornenie štruktúry hlavného balíčka modulu interpreter

Jednotlivé zložky balíčka sú rozdelené takto:

- **generated** balíček obsahuje vygenerovaný kód pomocou nástroja ANTLR. Obsahuje Lexer, Parser a Vistor, všetky vytvorené pre konkrétnu gramatiku, ktorú obsahuje súbor EasyLang.g4

- **visitor** balíček zoskupuje triedy, ktoré dedia od triedy EasyLangBaseVisitor. Tieto visitor triedy sa používajú na navštívenie vrcholov v prechádzanom derivačnom strome. Príkladom je trieda ArithmeticExpressionVisitor znázornená na diagrame (Obrázok 4.2)



Obrázok 4.2: Diagram triedy ArithmeticExpressionVisitor

- **enums** balíček v sebe obsahuje triedy enumerovaného typu

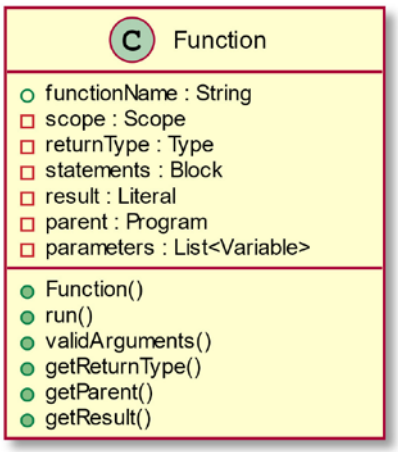
- **error** balíček obsahuje všetky triedy, ktoré sa starajú o chyby, ktoré vznikli počas interpretovania programu, či už počas lexikálnej alebo syntaktickej analýzy, alebo pri samotnom interpretovaní

- **util** balíček, podobne ako v module app, združuje pomocné triedy, medzi ktoré slúžia napríklad na vyhodnocovanie aritmetických výrazov, alebo na určovanie výsledného typu výrazov

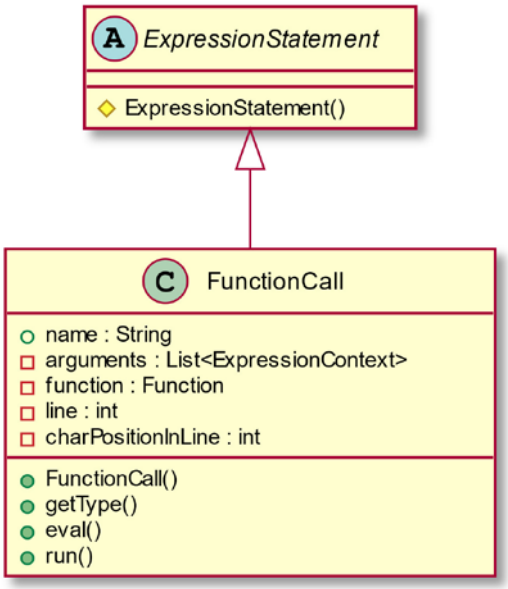
- **entity** balíček obsahuje triedy reprezentujúce neterminály v jazyku, v ktorých sú obsiahnuté všetky informácie, ktoré jednotlivé prvky jazyka potrebujú. Príklad takejto triedy je možné vidieť na diagrame (Obrázok 4.3: Diagram triedy Function). Tento balíček v sebe obsahuje ešte dva podbalíčky, kvôli prehľadnosti:

    - **expression** je balíček v ktorom sú uložené triedy implementujúce rozhranie Expression, ktoré umožňuje objekty týchto tried vyhodnocovať

- **statement** je balíček v ktorom sú zoskupené triedy dediace od abstraktnej triedy Statement, ktorá poskytuje možnosť spustiť daný príkaz. Zvláštnou triedou v tomto balíčku je abstraktná trieda ExpressionStatement, ktorej objekty majú vlastnosti výrazu, ktorý je taktiež spustiteľný ako príkaz. Táto trieda sú používa na vyjadrenie komponenty jazyka, ktorým je volanie funkcie – trieda FuncCall (Obrázok 4.4: Diagram triedy FuncCall).



Obrázok 4.3: Diagram triedy Function



Obrázok 4.4: Diagram triedy FuncCall

4.2 Jazyk

Pri vytváraní vlastného jazyka – nazvaného EasyLang – bol braný ohľad na jednoduchosť zápisov programov, ale taktiež na zachovanie podobnosti s konvenčnými programovacími jazykmi. Jazyk je interpretovaný do jazyka Java, z čoho vyplývajú tieto vlastnosti:

- **Staticky typovaný** – po vzore jazykov, akými je napríklad jazyk C, u ktorých platí, že typ výrazov je určený už počas kompilácie programu. Opačným príkladom sú dynamicky typované jazyky, u ktorých sa určujú typy výrazov až za behu programu.

- **Silne typovaný** – nepodporuje zaobchádzanie s hodnotami jedného typu ako hodnoty iného typu. V tomto však zdieľa rovnaké vlastnosti ako Java a preto sú povolené aj takéto zápisy, ktoré sú validné:

```
char A := 97;
int B := 'a';
if (A = B)
  write("true");
```

- **Imperatívny** – rovnako ako jazyk Pascal, je EasyLang prispôsobený človeku, a to kvôli tomu, že spôsob vykonávania programu je prirodzený pre človeka na rozdiel od iných druhov jazykov. Nie je však objektovo orientovaný, pričom ale postačuje na potreby výuky všetkých základných konceptov.

4.2.1 Gramatika

Syntax jazyka je definovaná gramatikou, ktorej popis nájdeme v súbore EasyLang.g4, prítomný v module interpreter. Gramatika je popísaná pravidlami vo formáte EBNF (Extended Backus–Naur form), ktorú využíva generátor ANTLR.

Syntaktické diagramy je možné nájsť v prílohe (A.2 Syntaktické diagramy). Na daných obrázkoch je znázornené, ako gramatika vyzerá a dáva nám návod na budovanie programov.

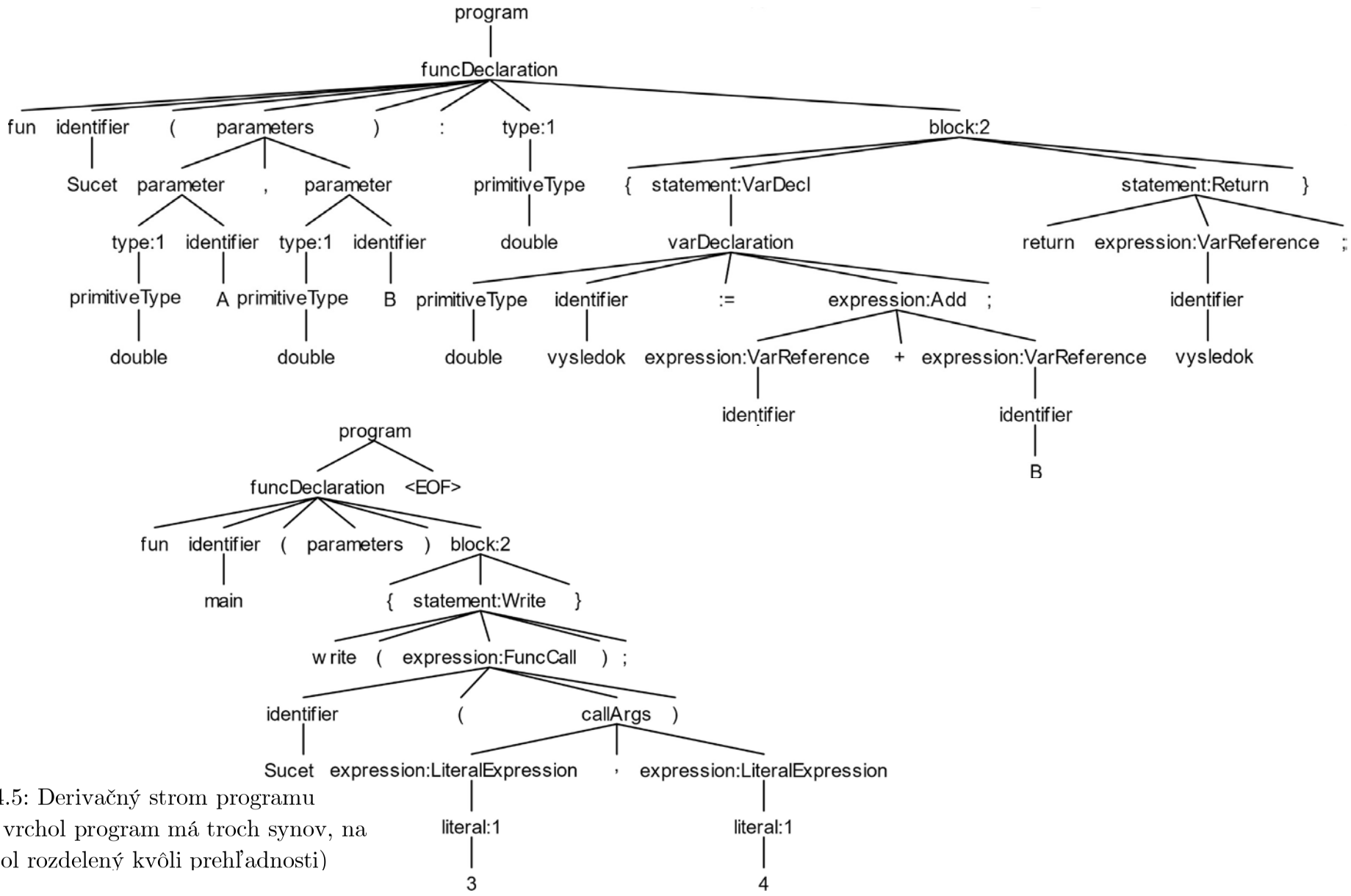
4.2.2 Príklad programu

Teraz sa pozrieme na konkrétny príklad programu, zapísaného v jazyku EasyLang. Príklad ukázaný v tejto kapitole je prevzatý z výukového textu 10 – Zhrnutie, ktorého je súčasťou. Príklad bol zároveň upravený tak, aby odpovedal potrebám tejto kapitoly.

```
fun Sucet(double A, double B) : double {
   double vysledok := A + B;
   return vysledok;
}

fun main() {
    write(Sucet(3,4));
}
```

Nástroj ANTLR po prijatí tohto kódu vygeneruje derivačný strom, po ktorom je následne možné prechádzať, jednotlivé vrcholy navštevovať a tým interpretovať kód. Derivačný strom je znázornený na obrázku (Obrázok 4.5: Derivačný strom programu).



Obrázok 4.5: Derivačný strom programu (pôvodný vrchol program má troch synov, na obrázku bol rozdelený kvôli prehľadnosti)

5. Spôsob výuky

V tejto kapitole si vysvetlíme, aké princípy boli zachovávané v rámci výuky, na aké časti bola výuka rozdelená a z akého dôvodu bolo zvolené poradie výuky konceptov

5.1 Všeobecné postupy

Celá výuka je rozdelená na menšie celky – koncepty. Jeden koncept obsahuje výukový text a voliteľne sadu testov slúžiacu na precvičovanie. Súbory konceptov sú dva – základné a pokročilé – kde súbor základných tvorí 10 konceptov a súbor pokročilých 9. Každý z výukových textov pozostáva z niekoľkých strán (v priemere 10 strán na jeden koncept); počet strán je pritom prispôsobený náročnosti danej témy. Testy sú rozdelené na dve časti, otázky s výberom možností a otázky s doplňovaním kódu. Všetky základné koncepty majú vlastnú sadu otázok s výberom možností, väčšina potom aj otázky s doplňovaním kódu. Pokročilé koncepty majú už iba otázky s výberom možností.

5.2 Základné koncepty

Prvým súborom sú základné koncepty, v rámci ktorých sa preberajú tieto témy (v určenom poradí):

- **Premenné I. (Príloha A.3)** – úvod do aplikácie; užívateľ sa zoznámi s prvým a zároveň najdôležitejším stavebným prvkom – premennými, ktoré sa budú používať v každom koncepte a preto musia byť vysvetlené na začiatku. Vysvetlenie je založené na príklade z reálneho sveta – krabičke. To uľahčuje užívateľovi tento koncept lepšie uchopiť. Taktiež sa užívateľ stretne s prvým programom – kalkulačkou na sčítavanie dvoch čísel.

- **Dátové typy I.** – po naučení sa premenných bolo potrebné predstaviť a vysvetliť základné primitívne typy, ktoré sa bežne v programovaní používajú. Vysvetľujú sa dátové typy integer, double, char a boolean, ktoré sú dostatočne jednoduché na pochopenie a preto by ich užívateľ mal byť schopný pochopiť bez problémov

- **Podmienky** – v tejto časti sa užívateľ stretne s podmienkami, ktoré dávajú význam dátovému typu boolean, ktorý bol vysvetlený v predošlej kapitole. Taktiež mu je prvýkrát predstavený vývojový diagram používaný ako súčasť znázornenia fungovania programu. Vysvetlené sú podmienky iba s pozitívnou vetvou, s pozitívnou aj negatívnou vetvou a dokonca aj vetvené podmienky. Tento koncept je vysvetlený na príklade porovnaní dvoch čísel, pričom sú na diagrame znázornené prechody vzhľadom k všetkým možným vstupom dvoch čísel.

- **Dátové typy II.** – druhá časť o dátových typoch ukazuje možnosti vytvorenia a použitia komplexných dátových typov, akými sú dátové typy string a pole. Vysvetlenie je postavené na príklade vlaku, ktorý v každom vagóne prevåža hodnotu rovnakého typu. To poskytuje užívateľovi predstavu, ako bežné pole v programe vyzerá. Pri type string je vysvetlený spôsob, akým tento typ vznikol a čím sa líši od typu char.

- **Cykly** – táto časť je, podobne ako časť Podmienky, celkom zložitá a preto sa na vysvetlenie používajú znova vývojové diagramy, ako spôsob na znázornenie behu programu. Vysvetlenie je postavené na prechádzanie poľom, pričom sa predpokladá, že užívateľ princíp poľa uchopil.

- **Funkcie I.** – celkový koncept funkcií je rozdelený na tri časti, pričom v prvej časti sa užívateľ zoznámi s funkciami, ktoré neprijímajú žiadne parametre a nevracajú výsledok. Koncept je vysvetlený na lieviku, kde v tejto časti je ukázaná prostredná časť – filter v lieviku – čo predstavuje kód funkcie. Príkladom je funkcia, ktorá vypíše pozdrav – Hello world!. Priebeh vykonávania programu je znovu znázornený a detailne popísaný na vývojovom diagrame.

- **Premenné II.** – po ukázaní si konceptu funkcií bolo nutné ukázať, ako funguje viditeľnosť premenných v programe. Užívateľ zistí, že k premenným vytvoreným vo vnútri funkcie nie je možné priamo pristupovať z vnútra inej funkcie. Ukázaný je taktiež spôsob vytvorenia globálnych premenných, stručne sú uvedené aj výhody a nevýhody použitia globálnych premenných

- **Funkcie II.** – v druhej časti vysvetľovania konceptu funkcií sú predstavené funkcie prijímajúce parametre. Princíp je vysvetlený na

funkcii Sucet(), prijímajúci dva parametre, ktorý na štandardný výstup vypisuje súčet týchto dvoch parametrov. Týmto sa doplnil príklad lievika, do ktorého sa niečo prilieva. V tejto kapitole je rovnako uvedený príklad, v ktorom je možné použiť ako globálne premenné, tak parametre a je vysvetlený dôvod prečo je lepšie použitie parametrov.

- **Funkcie III.** – posledná časť pojednávajúca o koncepte funkcií. V tejto kapitole je dokončený príklad na lieviku, zospodu ktorého môže niečo vychádzať. Použitý je rovnaký príklad ako v predošlej kapitole, pričom je funkcia Sucet() obohatená o návratový typ a návratovú hodnotu. Ďalej sú ukázané ostatné funkcie na vyhodnocovanie aritmetických výrazov, menovite Rozdiel(), Sucin() a Podiel(). Tieto funkcie budú slúžiť ako základ príkladu uvedeného v nasledujúcej kapitole – Zhrnutie.

- **Zhrnutie** – touto časťou je ukončená výuka základných konceptov v aplikácii. Slúži na ukázanie dvoch zložitejších príkladov, ktoré využívajú všetky koncepty, ktoré mal užívateľ doposiaľ možnosť uchopiť. Prvým príkladom je nájdenie maximálneho prvku v poli čísel. Príklad je postupne budovaný tak, aby užívateľ mal možnosť pochopiť jednotlivé časti postupne. Druhým príkladom je binárna kalkulačka, ktorá pozná štyri operácie – súčet, rozdiel, súčin a podiel. Oba príklady sú znázornené jednak na vývojovom diagrame a taktiež na kóde v programovacom jazyku.

5.3 Pokročilé koncepty

Druhým súborom sú základné koncepty, v rámci ktorých sa preberajú témy, ktoré sú vysvetľované bez príkladov na spustiteľnom kóde a za cieľ majú predstavenie a priblíženie pokročilejších nástrojov, princípov a komponent v konvenčných programovacích jazykoch.

V určenom poradí sa preberajú nasledujúce témy:

- **OOP I.** – úvod do objektovo orientovaného programovania. Užívateľovi poskytuje pohľad na iný spôsob programovania, než o akom sme pojednávali v základných konceptoch. Zoznámi sa s triedami a objektmi, ktoré sú neoddeliteľnou súčasťou množstva moderných programovacích jazykov. Vysvetlenie prebieha na príklade zo zvieracej ríše, kde znázorníme triedu Pes, z ktorej je možné vytvoriť objekty reprezentujúce konkrétnych psov. Užívateľ tak má možnosť nahliadnuť na myšlienku OOP.
- **OOP II.** – druhá časť o objektovo orientovanom programovaní prináša rozšírenie predošlej kapitoly o dedičnosť, ktorá je vysvetlená

na jednotlivých psích rasách. Ďalšou súčasťou tejto kapitoly je vysvetlenie konštruktorov ako nástroja na vytváranie nových objektov. Nakoniec sa užívateľ zoznámi s viditeľnosťou atribútov a metód a prečo je vhodné v určitých prípadoch obmedziť viditeľnosť

- **Algoritmy I.** – prvá časť popisujúca jednoduché algoritmy, s ktorými by sa užívateľ ako začiatočník mal zoznámiť a ktoré by sa mu mohli pri reálnom programovaní hodiť. Táto kapitola je zameraná na triediaci algoritmu Selection sort, ktorý je ľahko uchopiteľný pre začiatočníkov vzhľadom k tomu, že patrí medzi algoritmy blízke bežnému ľudskému premýšľaniu.

- **Algoritmy II.** – druhá kapitola o algoritmoch ukazuje ďalší triediaci algoritmus, ktorý je jednoduchý a často používaný začiatočníkmi vzhľadom k jednoduchosti zápisu v bežných programovacích jazykoch. Týmto algoritmom je Bubble sort a rovnako ako v predošlej kapitole, priebeh je pre lepšie pochopenie podrobne znázornený na konkrétnom príklade poľa.

- **Grafy I.** – kapitola, ktorá užívateľa vovádza do sveta grafov a stručne ukazuje teóriu grafov. Približuje rôzne typy grafov, akými sú orientované a neorientované, ohodnotené a neohodnotené a cyklické a acyklické grafy. Tieto grafy sú v stručnosti popísané a u niektorých je pridaný aj príklad z reálneho sveta, ktorý sa dá zaznamenať pomocou grafov.

- **Grafy II.** – v tejto kapitole sa spoločne s užívateľom pozrieme na mapu Slovenska, na ktorej sú vyznačené krajské mestá. K tomu je pripojená tabuľka vlakových spojov medzi každou dvojicou miest. Z týchto dát sa následne postaví graf, ktorý bude modelovať danú situáciu. Táto kapitola má ukázať, ako sa dá príklad z reálneho sveta previesť na graf.

- **Algoritmy III.** – posledná kapitola, v ktorej sa preberajú algoritmy bude zameraná na grafové algoritmy – prehľadávanie do šírky a prehľadávanie do hĺbky. Rovnako ako u iných algoritmov preberaných v týchto konceptoch sa neberie zreteľ na implementačné detaily, ale na to, aby bol užívateľ schopný znázorniť priebeh týchto algoritmov. Preto sú tieto algoritmy znázornené postupne, krok za krokom.

- **Grafy III.** - v poslednej kapitole o grafoch sa pozrieme na bludisko a budeme ho chcieť prejsť pomocou naučených algoritmov. Užívateľ

uvidí, ako je možné reprezentovať bludisko vo forme grafu a následne ho krok za krokom prejsť oboma naučenými algoritmami tak, aby našiel riešenie. V závere je znázornený väčší príklad bludiska, u ktorého si užívateľ, v prípade záujmu, môže vyskúšať prechod pomocou algoritmov.

- **Kolekcie** – kapitola, ktorá uzatvára výuku slúži na predstavenie kolekcií, ktoré sa bežne v programovaní využívajú. Medzi ne patrí zoznam, front a zásobník. U každej z týchto kolekcií sú uvedené príslušné vlastnosti a špecifiká, akými sú pridávanie a odoberanie prvkov z danej kolekcie. Každá táto operácia je znázornená na obrázku, čo zjednodušuje pochopenie.

6. Záver

Cieľom tejto práce bolo navrhnúť a vytvoriť mobilnú aplikáciu, ktorá slúži na výuku konceptov programovania spôsobom, ktorý by bol ľahko uchopiteľný pre užívateľov a zároveň im zjednodušil plynulý priebeh na nejaký z konvenčných programovacích jazykov. Pri budovaní aplikácie sa bral ohľad na jednoduchosť a intuitívnosť prostredia, v ktorom užívateľ nájde rýchlo a jednoducho všetko, čo potrebuje.

Tento princíp jednoduchosti bol v istej miere implementovaný aj do výuky, ktorá prebieha spôsobom, ktorý by mal byť jednoduchšie pochopiteľný vzhľadom napríklad k paralelám s reálnym svetom, ktorý užívateľ pozná. Výuka zachovávala princíp postupného budovania vedomostí a znalostí a každý nový koncept sa predstavuje po častiach tak, aby užívateľ bol čo najmenej zahltený veľkým množstvom nových informácií.

Výuka je taktiež doplnená testami, ktoré umožňujú užívateľovi overiť si nadobudnuté znalosti na testoch s výberom správnych možností, respektíve s doplňovaním kódu.

Ďalšou súčasťou aplikácie je vstavaný interpreter vlastného programovacieho jazyka EasyLang, ktorý má za úlohu spúšťať kód a zároveň pomáha pri vyhodnocovaní testov s doplňovaním kódu. Jazyk bol vytváraný tak, aby syntax bola jednoduchá na pochopenie, respektíve aby všetky súčasti boli zároveň jednoduché a zároveň aby sa príliš nelíšili od bežných, konvenčných programovacích jazykov, ktoré sa v dnešnej dobe používajú.

Testovanie aplikácie prebiehalo najmä na mobilnom telefóne s Androidom 6.0.1 (API 23), pričom ale sekundárne sa testovalo v emulátore Android Studia na emulátoroch mobilných zariadení s verziou Androidu 4.4, 5.0 a 8.1. Toto testovanie by malo minimalizovať počet chýb na reálnych zariadeniach.

Aplikácia by sa do budúcnosti mohla rozširovať do troch rovín. Na jednej strane sú nové výukové texty, ktoré by niektoré existujúce koncepty mohli ďalej rozšíriť, prípadne ktoré by predstavovali úplne nové koncepty, ktoré do súčasnej výuky nie sú zahrnuté. Druhou rovinou je interpreter, ktorý nepodporuje objektovo orientované programovanie. Po pridaní tejto súčasti do jazyka by bolo možné texty pokročilých konceptov rozšíriť o názorné príklady programov zapísané v tomto programovacom jazyku. Poslednou rovinou je samotná aplikácia, ktorá by mohla ešte interaktívnejšie komunikovať s užívateľom, pridaný by mohol byť časový plán výuky, v ktorom by v každý zadaný čas užívateľovi prišla notifikácia o tom, aby sa naučil nový koncept, respektíve si skúsil nové testy.

Zoznam použitej literatúry

[1] **Parys, Jonathan Van. 2012.** The most spoken languages in the European Union. *Language Knowledge.* [Online] 2012. [Dátum: 25. 04 2019.] https://languageknowledge.eu/countries/slovakia.

[2] **Kalaš, Ivan. 2017.** Výučba informatiky na 1. stupni ZŠ. *Quark.* [Online] 2017. [Dátum: 25. 04 2019.] http://www.quark.sk/vyucba-informatiky-na-1-stupni-zs/.

[3] **StatCounter. 2019.** Mobile & Tablet Operating System Market Share Worldwide - March 2019. *GlobalStats StatCounter.* [Online] StatCounter, 2019. [Dátum: 25. 04 2019.] http://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/#yearly-2012-2019.

[4] **Google Developers.** App manifest file. *Guide.* [Online] Google Developers. [Dátum: 25. 04 2019.] https://developer.android.com/guide/topics/manifest/uses-sdk-element.

[5] **Google Developers. 2018.** Distribution dashboard. *Platform.* [Online] Google Developers, 2018. [Dátum: 25. 4 2019.] https://developer.android.com/about/dashboards/.

[6] **Jetbrains. 2016.** Kotlin 1.0 Released. *Kotlin Web.* [Online] Jetbrains, 15. 02 2016. [Dátum: 02. 05 2019.] https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/.

[7] **Jetbrains. 2017.** Kotlin blog. *Kotlin on Android. Now official.* [Online] Jetbrains, 17. 05 2017. [Dátum: 02. 05 2019.] https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/.

[8] **Eremin, Ilya. 2017.** Android JSON Parsers Comparison. *Medium.* [Online] Medium, 05. 03 2017. [Dátum: 02. 05 2019.] https://medium.com/@IlyaEremin/android-json-parsers-comparison-2017-8b5221721e31.

[9] **Parr, Terence.** About: ANTLR. *ANTLR.* [Online] [Dátum: 02. 05 2019.] https://www.antlr.org/about.html.

[10] **Android Pit. 2012.** Holo UI – Google Plans Uniform Design For All Android 4.0 Devices. *Android Pit.* [Online] Android Pit, 04. 01 2012. [Dátum: 02. 05 2019.] https://www.androidpit.com/Holo-UI-Google-Plans-Uniform-Design-For-All-Android-4-0-Devices.

[11] **Google. 2014.** Google. *Google I/O 2014 - Material design principles.* [Online] Google, 25. 06 2014. [Dátum: 02. 05 2019.] https://www.google.com/events/io/io14videos/79edef8b-96d4-e311-b297-00155d5066d7.

[12] **Intuit.** *Github.* [Online] Github. [Dátum: 02. 05 2019.] https://github.com/intuit/sdp.

[13] **Schiller, Bartosz. 2017.** *Github.* [Online] Github, 2017. [Dátum: 02. 05 2019.] https://github.com/barteksc/AndroidPdfViewer.

[14] **Google Developers.** *Layouts.* [Online] Google. [Cited: 05 10, 2019.] https://developer.android.com/guide/topics/ui/declaring-layout.

A. Prílohy

A.1 Vstupný JSON formát

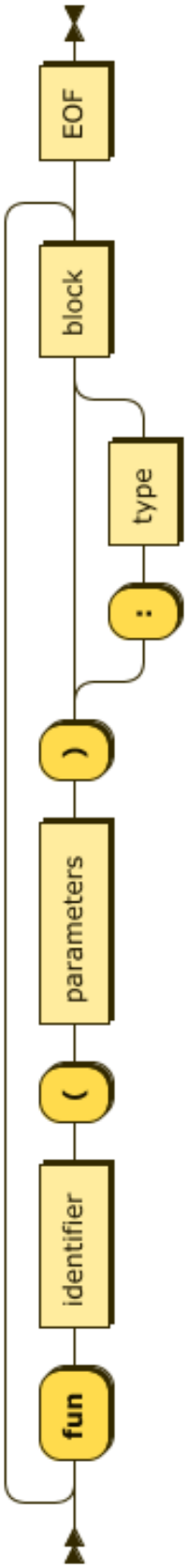
```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "level": {
        "type": "integer",
        "examples": [
          0
        ]
      },
      "name": {
        "type": "string",
        "examples": [
          "Premenné"
        ]
      },

      "description": {
        "type": "string",
        "examples": [
          "Základný koncept, ktorý to celé odštartuje"
        ]
      },
      "conceptDrawable": {
        "type": "string",
        "examples": [
          "ic_variable1"
        ]
      },
      "numTestScreens": {
        "type": "integer",
        "examples": [
          5
        ]
      },
      "fileName": {
        "type": "string",
        "examples": [
          "Premenné I.pdf"
        ]
      },
      "choicesQuestions": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "descriptionText": {
              "type": "string",
              "examples": [
                "Aký symbol sa používa na uloženie do premennej?"
              ]
```

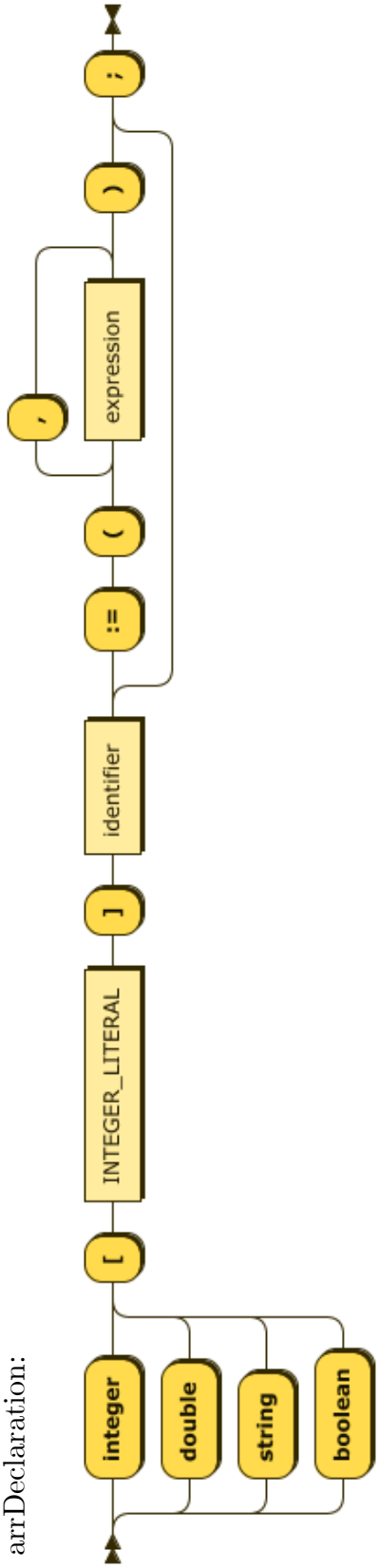
```
            },
            "choices": {
              "type": "array",
              "items": {
                "type": "string",
                "default": "",
                "examples": [
                  "<-",
                  "=",
                  ":=",
                  "=:"
                ]
              } },
            "correctAnswerIndices": {
              "type": "array",
              "items": {
                "type": "integer",
                "default": 0,
                "examples": [
                  2
                ]
              } }
          } }
        } },
      "codeQuestions": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "description": {
              "type": "string",
              "examples": [
                "Vypíšte premennú A"
              ]
            },
            "code1": {
              "type": "string",
              "examples": [
                "integer A := 10;"
              ]
            },
            "code2": {
              "type": "string",
              "examples": [
                ""
              ]
            },
            "test": {
              "type": "object",
              "properties": {
                "output": {
                  "type": "string",
                  "examples": [
                    "10"
                  ]
                } }
            } }
        } }
    } }
}
```

A.2 Syntaktické diagramy

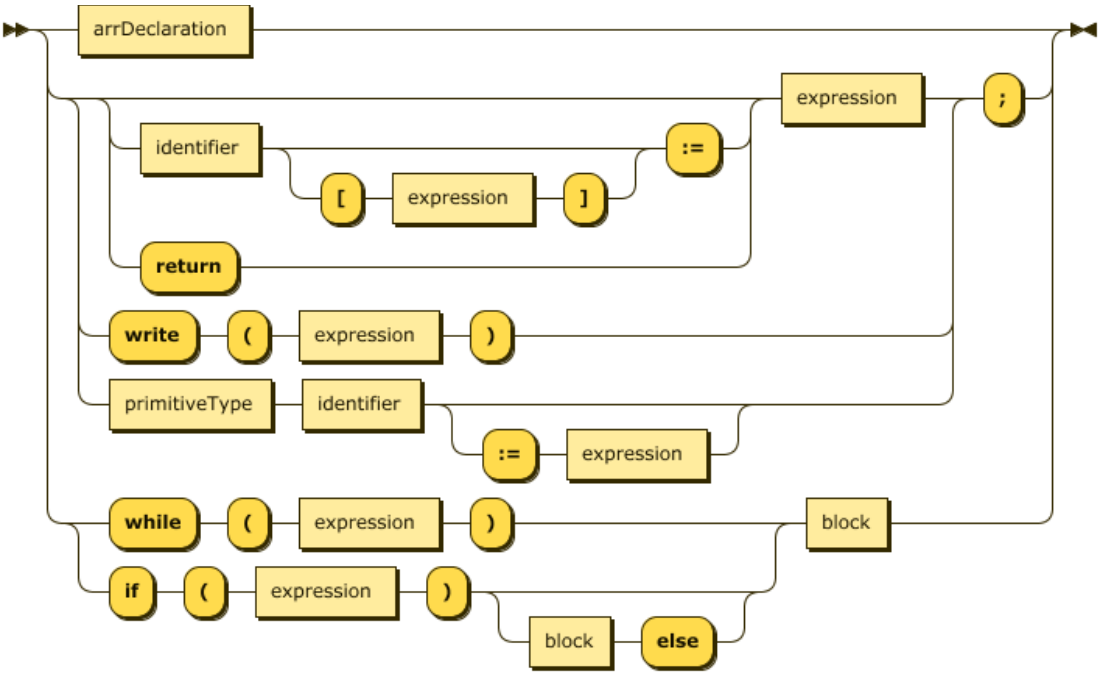
program:



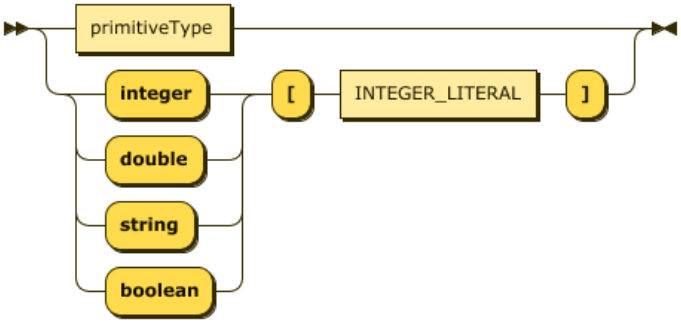
arrDeclaration:



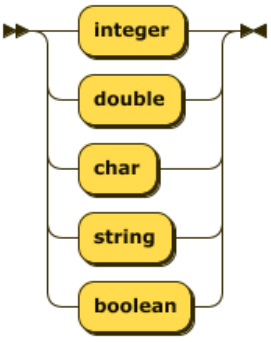
statement:



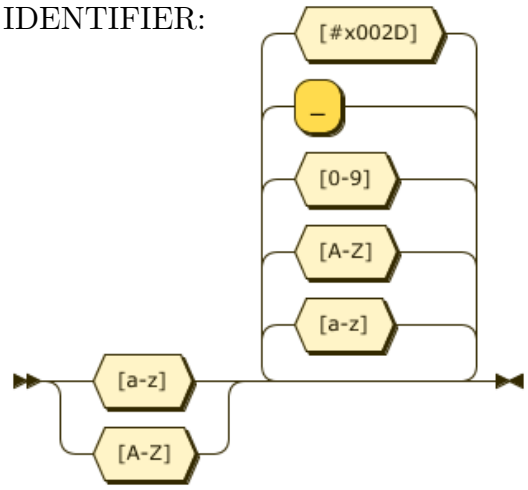
type:



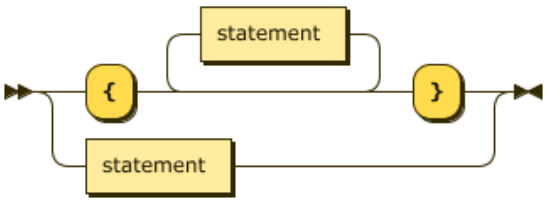
primitiveType:



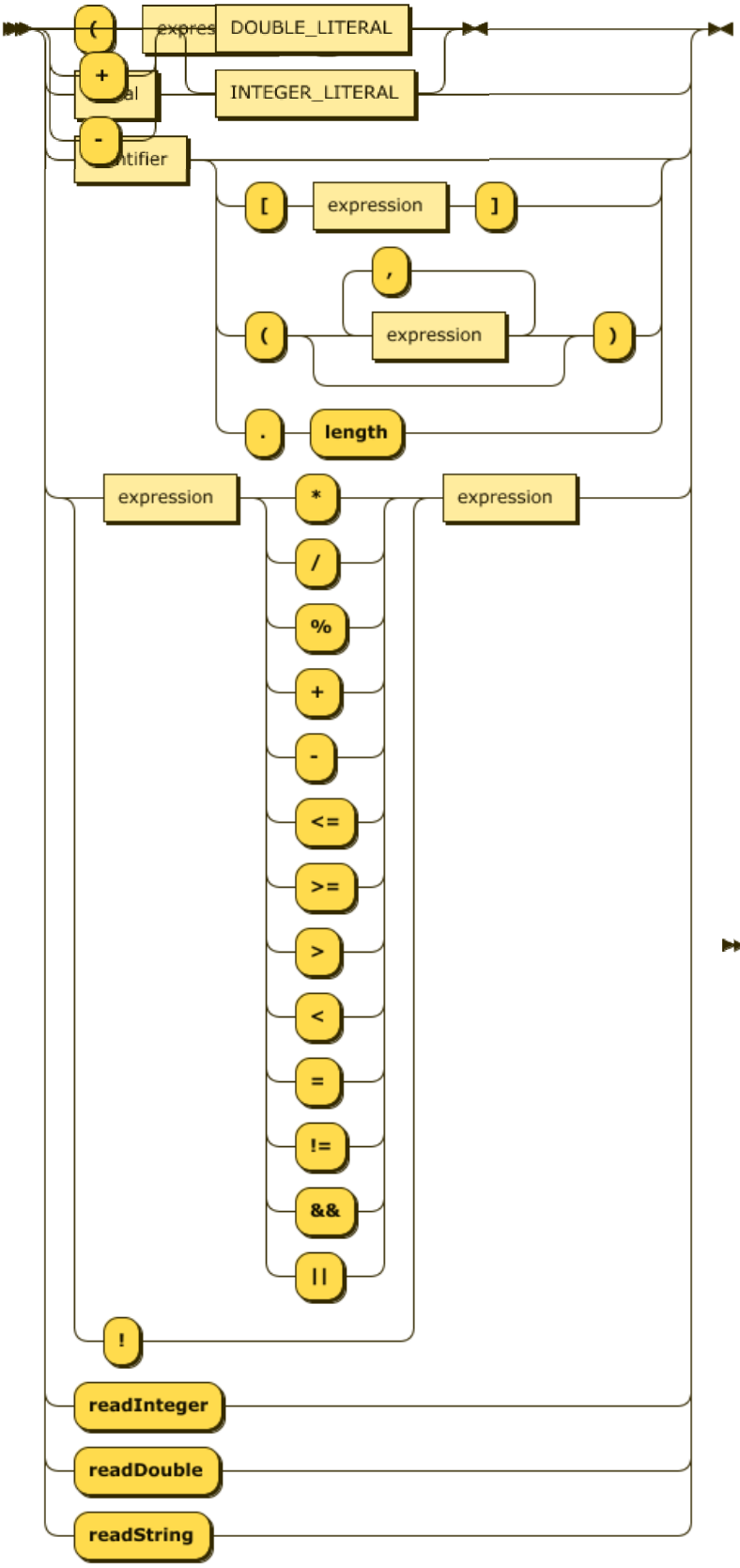
IDENTIFIER:



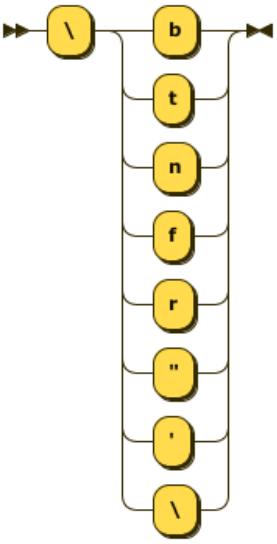
block:



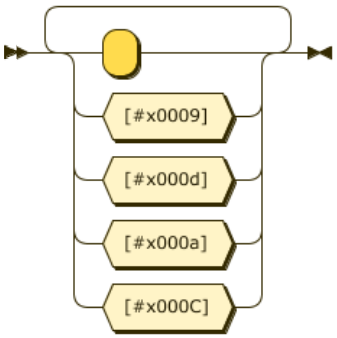
number:



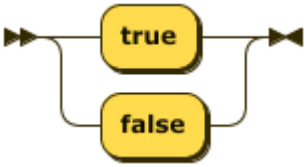
Escape:



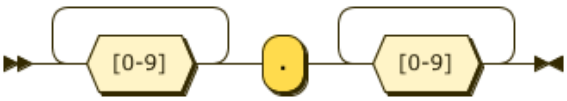
WHITESPACE:



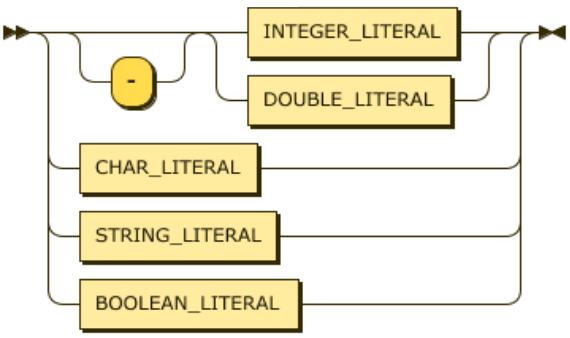
BOOLEAN_LITERAL:



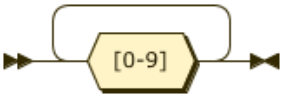
DOUBLE_LITERAL:



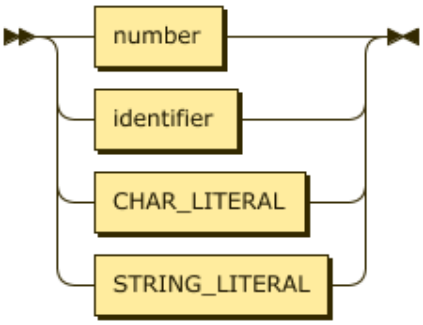
WHITESPACE:



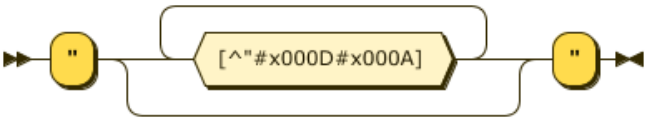
INTEGER_LITERAL:



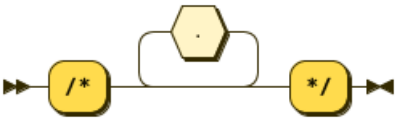
WHITESPACE:



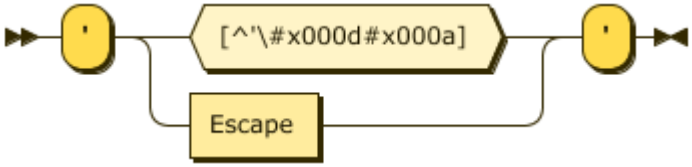
STRING_LITERAL:



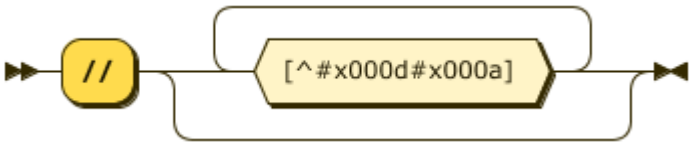
COMMENT:



CHAR_LITERAL:



LINE_COMMENT:



A.3 Príklad výukového textu

PREMENNÉ

1. ÚVOD

ČO JE PREMENNÁ?
NA ČO JE V PROGRAME POTREBA?

Pri vytváraní programov a ich používaní je potrebné mať nejaký prostriedok na to, aby naše programy boli univerzálne.

Ukážeme si to na príklade kalkulačky, pomocou ktorej budeme sčítavať dve čísla. Užívateľ takejto kalkulačky zadá najprv jedno číslo, potom druhé. Následne mu kalkulačka vypočíta výsledok.

Takýto program sme schopní napísať celkom jednoducho, ak máme premenné - krabičky, do ktorých si ukladáme čísla, ktoré sme dostali od užívateľa.

PREMENNÉ

2. KRABIČKA

PREČO KRABIČKA?
ČO SA DÁ DO KRABIČKY VLOŽIŤ?

Premenné pochopíme najlepšie, keď si predstavíme, že sa chceme sťahovať a potrebujeme všetky naše veci zbaliť do krabíc, aby sme si ich potom mohli vybrať a znovu uložiť na správne miesto v novom byte.

Je viac druhov vecí, ktoré potrebujeme zbaliť, je to napríklad oblečenie, kuchynské zariadenie, alebo taktiež dokumenty. Každý takýto druh bude mať vlastnú krabicu, aby sa nám to nepoplietlo.

S premennými to je podobné, do jednej si uložíme celé čísla, do druhej desatinné čísla a do ďalšej zase textové reťazce. Tieto druhy sa volajú dátové typy a budeme sa im venovať v ďalšej kapitole.

Aby sme mohli pokračovať v spoznávaní premenných, bude nám stačiť poznať zatiaľ dátový typ celých čísel - integer.

4. POUŽÍVANIE

AKO VYBRAŤ A POUŽIŤ ČÍSLO Z PREMENNEJ?

Teraz sa presunieme k ďalšej časti našej kalkulačky a tou je spočítanie zadaných čísel. Teraz je nejak vybrať hodnoty, ktoré sú v premenných A a B a spočítať ich.

Na to, aby sme použili hodnotu, ktorá je v premennej uložená, stačí použiť názov premennej.

Vytvorím novú premennú, pomenovanú Sum, do ktorej uložím súčet A a B. V našom jazyku sa to môže zapísať následovne:

```
integer Sum;
Sum := A + B;
```

3. UKLADANIE

AKO ULOŽIŤ ČÍSLO DO PREMENNEJ?

Vráťme sa k príkladu kalkulačky. Chceli sme sčítať dve čísla, ktoré, ako sme zistili, je najlepšie mať uložené v premenných.

Teraz musíme vyriešiť problém, ako číslo do premennej uložiť. Každá premenná bude v sebe obsahovať jedno číslo. Premenné si pomenujeme A a B, do A uložím číslo 5, do B číslo 3.

V rôznych programovacích jazykoch je možné uložiť hodnotu do premennej inak; v nami používanom jazyku to urobíme takto:

```
integer A := 5;
integer B := 3;
```

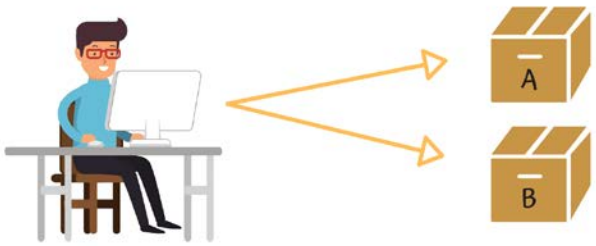
6. VSTUP A VÝSTUP

AKO ZÍSKAŤ OD UŽÍVATEĽA ČÍSLA?
AKO ZOBRAZIŤ UŽÍVATEĽOVI VÝSLEDOK?

Skúsme teraz vniesť kúsok dynamiky do našej kalkulačky. Nechceme stále spúšťať program, ktorý bude počítať iba čísla 5 a 3, taký program by nebol veľmi užitočný.

Dajme užívateľovi možnosť vybrať si, aké čísla chce sčítať. Na to sa v programovacích jazykoch používa čítanie vstupu, u nás na to existuje výraz readInteger, ktorý slúži na čítanie celých čísel.

```
integer A := readInteger;
integer B := readInteger;
```



5. ZMENA

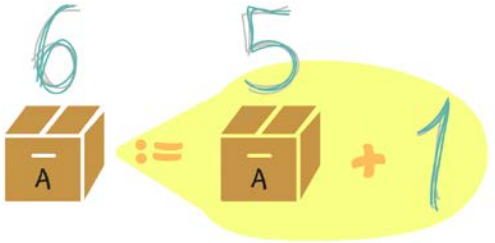
AKO ZMENIŤ ČÍSLO V PREMENNEJ?

Ukážme si teraz niečo, čo sa nám môže v budúcnosti hodiť a čo budeme často používať. Vezmime si premennú A v ktorej je uložené číslo 5.

Chceli by sme k tomuto číslu pričítať číslo 10, ale nechceme na to vytvárať novú premennú. V takom prípade využijeme operátor priradzovania := takto:

```
integer A := 5;
A := A + 1;
```

A ako funguje takto uvedený kód? Operátor := si môžeme predstaviť ako šípku, ktorá ukazuje vľavo, čo znamená, že sa najprv vyhodnotí pravá strana a uloží sa premennej na ľavej strane.



6. VSTUP A VÝSTUP

AKO ZÍSKAŤ OD UŽÍVATEĽA ČÍSLA?
AKO ZOBRAZIŤ UŽÍVATEĽOVI VÝSLEDOK?

Vstupné čísla máme, vieme aj ako spočítať súčet čísel, ktorý sme si uložili do premennej Sum. Teraz nám ostáva poskytnúť užívateľovi nejakú odozvu o výpočte.

Na tieto účely sa používa funkcia write, ktorej zadáme to, čo sa má vypísať. Vypíšeme pomocou nášho jazyka premennú Sum:

```
integer Sum := A + B;
write(Sum);
```

7. KALKULAČKA

AKO SI NAPÍŠEME VLASTNÚ KALKULAČKU?
AKO FUNGUJE TENTO PROGRAM?

Zhrňme si všetko, čo sme sa naučili. Vieme vytvoriť premennú, vieme do nej uložiť vstup od užívateľa, vieme spočítať uložené hodnoty v premenných a výsledok zobraziť užívateľovi.

Ukážeme si to na programe:

```
integer A := readInteger;
integer B := readInteger;
integer Sum := A + B;
write(Sum);
```

Na prvých dvoch riadkoch sme vytvorili premenné A a B, ktoré sú typu integer, teda celých čísel a zároveň sme do nich vložili vstup od užívateľa.

Tretí riadok vytvorí premennú Sum a uloží do nej súčet A a B.

Štvrý riadok vypíše výsledok uvedený v Sum.

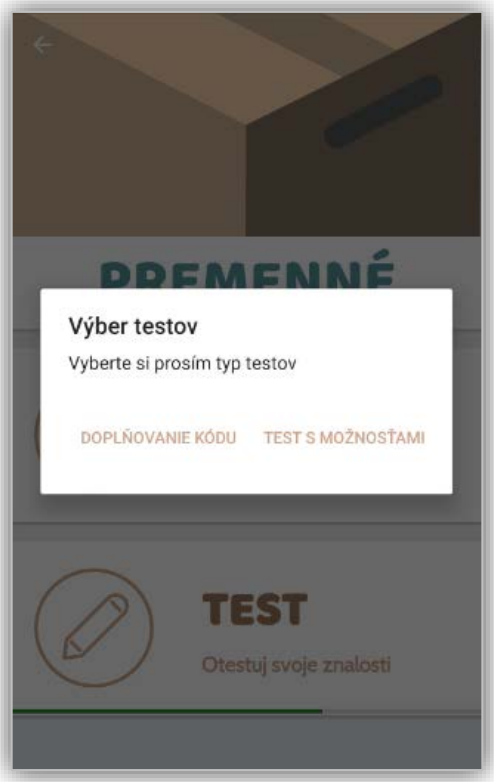
HOTOVO! Vytvorili sme náš prvý program.

B. Užívateľská dokumentácia

Aplikácia pozostáva z niekoľkých obrazoviek. Po spustení sa zobrazí obrazovka so zoznamom konceptov.

Po výbere položky v zozname sa užívateľ presunie na ďalšiu obrazovku s detailom o vybranom koncepte. Na tejto obrazovke je zobrazený taktiež postup v učení sa daného konceptu. Po prejdení celej výuky sa zobrazí značka ✓, ktorá symbolizuje zvládnutie výuky. Rovnako tak na karte test je zaznamenaný zeleným pásikom dosiahnutý postup v teste.

Po kliknutí na položku výuka v obrazovke detailu konceptu sa presunieme do obrazovky s výukou. Text je možné na tejto obrazovke približovať a posúvať sa medzi obrazovkami rovnako tak, ako je to bežné v podobných aplikáciách. Späť z tejto obrazovky sa užívateľ dostane stlačením tlačidla späť na svojom zariadení.



V prípade, že užívateľ klikne na položku test v obrazovke detailu, ukáže sa dialógové okno, ktoré ponúkne výber testov. Užívateľ má dve možnosti.

Ak vyberie možnosť s dopĺňaním kódu, zobrazí sa mu obrazovka, na ktorej je pripravené pole určené na vloženie kódu. Po kliknutí na tlačidlo Vyskúšať, užívateľ môže vyskúšať spustiť kód, pričom sa mu zobrazí dialógové okno s výstupom. Pri kliknutí na šípku sa potom kód vyhodnotí a v prípade, že je odpoveď správna, prejde sa na ďalšiu otázku, alebo sa ukončí test.

V opačnom prípade, po výbere položky Test s možnosťami, sa užívateľ presunie na obrazovku, aká je znázornená na obrázku. Užívateľ si prečíta otázku a vyberie jednu alebo viac správnych odpovedí. Potom klikne na šípku, pomocou ktorej sa odpovede vyhodnotia, podá sa informácia o správnosti odpovede a následne sa prejde na ďalšiu otázku, prípadne na ukončenie testu s vyhodnotením.

Poslednou súčasťou aplikácie je takzvané Pieskovisko a dostaneme sa doň po kliknutí vpravo hore na hlavnej obrazovke. Tento nástroj ponúka možnosť vyskúšať si napísať nejaký program a zistiť o našom programovacom jazyku trochu viac. Po kliknutí znova do pravého rohu na otáznik sa zobrazí príručka syntaxe jazyka.